(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
19 February 2004 (19.02.2004)

PCT

(10) International Publication Number
WO 2004/015572 A1

(51) International Patent Classification[7]: G06F 9/50, 12/00

(21) International Application Number:
PCT/AU2003/000994

(22) International Filing Date: 6 August 2003 (06.08.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/401,886	7 August 2002 (07.08.2002)	US
60/402,024	7 August 2002 (07.08.2002)	US
60/402,013	7 August 2002 (07.08.2002)	US
60/420,181	22 October 2002 (22.10.2002)	US
60/420,486	22 October 2002 (22.10.2002)	US

(71) Applicant *(for all designated States except US)*: **MMAGIX TECHNOLOGY LIMITED** [GB/GB]; 4 Upper Church Street, Douglas, Isle of Man IM1 1EE (GB).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **O'SULLIVAN, Daniel, Shane** [AU/AU]; 38 Bateman Road, Mt Pleasant, Western Australia 6153 (AU).

(74) Agent: **STARKIE, Steven, John**; Griffith Hack Patent Attorneys, 256 Adelaide Terrace, PERTH, Western Australia 6000 (AU).

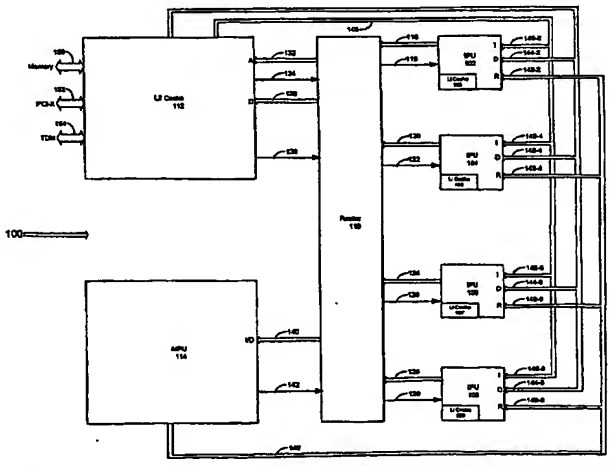
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) Title: APPARATUS, METHOD AND SYSTEM FOR A SYNCHRONICITY INDEPENDENT, RESOURCE DELEGATING, POWER AND INSTRUCTION OPTIMIZING PROCESSOR



(57) Abstract: An apparatus, method, and system for synchronicity independent, resource delegating, power and instruction optimizing processor is provided where instructions are delegated between various processing resources of the processor. An Integer Processing Unit (IPU) of the processor delegates complicated mathematical instructions to a Mathematical Processing Unit (MPU) of the processor. Furthermore, the processor puts underutilized processing resources to sleep thereby increasing power usage efficiency. A cache of the processor is also capable of accepting delegated operations from the IPU. As such, the cache performs various logical operations on delegated requests allowing it to lock and share memory without requiring extra processing cycles by the entire processor. With the processor, execution instructions are optimized reducing the complexity of the processor, throughput is increased as delegation to multiple processing resources is scalable, and power usage efficacy is increased as underutilized and/or waiting processing resources may sleep when not active.

WO 2004/015572 A1

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

APPARATUS, METHOD AND SYSTEM FOR A SYNCHRONICITY INDEPENDENT, RESOURCE DELEGATING, POWER AND INSTRUCTION OPTIMIZING PROCESSOR

FIELD

The present invention relates generally to an apparatus, method and system of processor architecture. More particularly, the disclosed invention(s) relate to an apparatus, method and system of delegating and optimizing the issuance and execution of execution-instruction signals among multiple intra-processing resources.

The present application claims priority from US 60/401,886, US 60/402,013, US 60/402,024, US 60/420,486 and US 60/420,181 and the disclosures in these documents are incorporated herein by reference.

BACKGROUND

Typically, users, which may be people or other systems, engage information technology systems (e.g., commonly computers) to facilitate information processing. In turn, computers employ processors to process information and such processors are often referred to as central processing units (CPUs). A common form of processor is referred to as a microprocessor. A computer operating system, which is software that is executed on a computer, enables and facilitates users to access and operate computer information technology and resources. Common resources employed in information technology systems include input and output mechanisms through which data may pass, memory storage into which data may be saved, and processors by which information may be processed. Often information technology systems are used to collect data for later retrieval, analysis, and manipulation, which often is facilitated through database software. Information technology systems provide interfaces that allow users to access and operate various system components.

Early single chip microprocessors included the Intel 8080 (an 8-bit processor introduced in 1974), the Motorola 6502 (an 8-bit processor introduced in 1977), the Intel

8088 (introduced in 1979 and incorporated into the IBM Personal Computer (PC)). The vast majority of the PC market moved from the Intel 8088 to the 80286 to the 80386 to the 80486 to the Pentium to the Pentium II to the Pentium III to the Pentium 4. While the current microprocessor market is dominated by the Intel architecture, other processors such as the Motorola 68000 and PowerPC have minor market shares.

The various processors employ disparate architectures including a complex instruction set computer (CISC) and a reduced instruction set computer (RISC). Microprocessors provide instruction sets that allow program developers to exploit and use the microprocessors to execute and process information. Typically, program developers use program languages that are compiled into a microprocessor's instruction set by a special program called a compiler. Compilers themselves are made by program developers who use tools to design compilers to produce instructions for a specific microprocessor.

These designs of the microprocessor originated several years ago, and have evolved over time. As the designs have evolved, the clock rate of the microprocessor has been increased to deal with more instructions within a fixed time frame.

SUMMARY OF THE INVENTION

In accordance with non-limiting, present, inventive aspects of the disclosure, problems of increased power requirements, heat generation, design complexity and resulting reduced processing efficiency in processors are overcome.

The increased clock rate of microprocessor architectures in the prior art causes many serious problems such as deepened pipelines, increased power requirements, which result in elevated heat in the chip, all of which making the already complex circuitry even more complicated. Further, as the designs have evolved, it has been necessary to maintain compatibility with earlier products. This results in a large number of hardware features being incorporated that are not used in new application software because the new software tends to comply with existing hardware features. This represents a waste of hardware capacity that could be better employed.

The above-identified problems are solved and a technical advance is achieved in the art of a processor with a synchronicity independent, power and instruction optimizing, and resource delegating processor of the present invention. An exemplary processor of the present invention includes: an integer processing unit adapted to execute instructions of a program; a mathematical processing unit adapted to receive a request from the integer processing unit; and a router that interfaces between the integer processing unit and the mathematical processing unit, wherein the router is adapted to route a request from the integer processing unit to the mathematical processing unit.

In accordance with another, non-limiting, present, inventive aspect of the disclosure, there is provided an architecture of a processor for executing instructions of a program. An exemplary processor of the present invention includes: a plurality of processing units each capable of executing instructions of a program independently; and a cache unit configured to receive access requests from the plurality of processing units and returns results to the plurality of processing units in response to the access requests, wherein each of the plurality of processing units is configured to sleep after sending an access request to the cache unit.

In accordance with another, non-limiting, present, inventive aspect of the disclosure, there is provided an architecture of a processor for executing instructions of a program. An exemplary processor of the present invention includes: a processing unit with an internal cache unit, residing internal to the processing unit, configured to execute instructions of a program; and an external cache unit, residing external to the processing unit, configured to cooperate with the internal cache unit of the processing unit, wherein each of the internal and external cache units is divided into an instruction area and a data area, and the data area is further subdivided into a local area and a global area thereby increasing the cache hit rate.

The above advantages and features are of representative embodiments only, and are not exhaustive and/or exclusive. They are presented only to assist in understanding the invention. It should be understood that they are not representative of all the inventions defined by the claims, to be considered limitations on the invention as defined by the claims, or limitations on equivalents to the claims. For instance, some of these advantages may be

mutually contradictory, in that they cannot be simultaneously present in a single embodiment. Similarly, some advantages are applicable to one aspect of the invention, and inapplicable to others. Furthermore, certain aspects of the claimed invention have not been discussed herein. However, no inference should be drawn regarding those discussed herein relative to those not discussed herein other than for purposes of space and reducing repetition. Thus, this summary of features and advantages should not be considered dispositive in determining equivalence. Additional features and advantages of the invention will become apparent in the following description, from the drawings, and from the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings illustrate various non-limiting, example, inventive aspects of the present disclosure:

Figs. 1A, 1B, 1C are of diagrams illustrating non-limiting example, present, inventive aspects of the disclosure providing overviews of a data processing system;

Fig. 2 is of a functional block diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing an architecture of a processor;

Fig. 3 is of a simplified schematic diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of the data paths of integer processing unit requests from a router to a target processing resource of a processor;

Fig. 4 is of a simplified schematic diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of the data path of an integer processing unit of a processor;

Fig. 5 is of a logic diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of the hashing function which is connected to each of the outputs of integer processing units;

Fig. 6 is of a simplified schematic diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of an instruction decoder of an integer processing unit of a processor;

Fig. 7 is of a simplified schematic diagram illustrating an alternative, non-limiting example, inventive aspect of the present disclosure of the data path of an integer processing unit of a processor;

Fig. 8 is of a simplified schematic diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of the instruction/data path of an external level two cache of a processor;

Fig. 9 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a resource delegation process in an integer processing unit in which the integer processing unit constructs and delegates a request to other processing resources;

Fig. 10 is of a block diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a form of a delegation request constructed at a requesting integer processing unit and directed to a mathematical processing unit;

Fig. 11 is of a block diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a form of a delegation request constructed at a requesting integer processing unit and directed to an external level two cache of a processor;

Fig. 12 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of request handling in a router of a processor routing requests between the requesting processing resources and servicing processing resources;

Fig. 13 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of a response from a target processing resource to the requesting processing resource;

Fig. 14 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of routing a result from a target processing resource to delegating processing resources;

Fig. 15 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a sleep lock operation in an external cache of a processor;

Fig. 16 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of maintaining the cache coherence between the internal caches and the external cache of a processor;

Fig. 17 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of a data optimizing cache facility between the internal and external caches of a processor;

Figs. 18(a)~(e) are of functional block diagrams illustrating non-limiting example, inventive aspects of the present disclosure providing a format of instructions designed for a processor;

Fig. 19 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of literal prefix and literal constant functionality;

Fig. 20 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a late/dynamic binding accelerator;

Fig. 21 is of a table illustrating one, non-limiting example, inventive aspect of the present disclosure of multicycle instruction reduction;

Fig. 22 is of a table illustrating one, non-limiting example, inventive aspect of the present disclosure which provides various integer processing unit registers, which may be accessed based on type flags that are set in a reduced register address portion of an instruction;

Fig. 23 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing an event sleep facility;

Fig. 24 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing an event set instruction facility;

Fig. 25 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a wait-on-semaphore instruction facility;

Fig. 26 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a signal-on-semaphore instruction facility.

Fig. 27 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a sleep lock instruction facility;

Fig. 28 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing an unlock operation;

Fig. 29 continues to illustrate the unlock operation from Fig. 28; and

Fig. 30 continues to illustrate an unlock operation in ensuring the cache coherence of Fig. 29.

DETAILED DESCRIPTION

In the following description of the various embodiments, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration various embodiments in which the invention may be practiced. It is to be understood that other embodiments may be utilized and structural and functional modifications may be made without departing from the scope of the present invention.

Data processing System Overview

Figs. 1A, 1B, 1C are of diagrams illustrating non-limiting example, present, inventive aspects of the disclosure providing overviews of a data processing system 10 in which a Processor 100 of the present invention is connected to external devices. It should be noted that these overviews are reference systems and that Processor 100 may be deployed in numerous contexts such as, but not limited to, communications systems, desktop systems, embedded systems, extra-terrestrial systems, server systems, and/or the like. Briefly, Processor 100 may be configured to connect directly to at least one of external memories 22, 24, 26, 28, a bus 20, and an input/output (I/O) bus 30 to result in a data processing system. The I/O bus may employ any number of bus architectures such as, but not limited to accelerated graphics port (AGP), card bus, (extended) industry standard architecture (EISA), micro channel architecture (MCA), NuBus, peripheral component interconnect (extended) (PCI-X), personal computer memory card international association (PCMCIA), and/or the like.

In one non-limiting example embodiment 2 of Fig. 1A, a synchronicity independent, resource delegating, power and instruction optimizing Processor 100 of the present invention is provided with an external memory interface 12 which may be configured to be connected to up to 4 GB (~2^{32} bits) of memory built with various memory technologies and/or capacities 22, 24, 26, 28. In alternative embodiments, Processor 100 may employ greater than 32-bits for addressing and/or processing information, and as such may address more memory (e.g., 48-bits, 64-bits, 128-bits, etc.). Fig. 7, for example, illustrates a non-limiting 64-bit example embodiment of Processor 100 addressing. The external memory interface may support system bus 20 with 72-bit parallel transfers, providing 64-bit words with error control. Each of the memories may be segmented into smaller, say 64 KB segments and larger, say 1 MB segments closely matching the memory allocation and management methods of multi-user operating systems such as Unix. Each of the memory segments may be locked to a thread ID or with a shared group lock for an improved utilization efficiency of the memories. Instead of using the virtual memory mapping as in conventional processor technologies, which takes extra time for accesses, Processor 100 of the present invention uses real memory as required by applications. Using the real memory improves the quality of multimedia data access and presentation, which require real-time processing. Alternatively, virtual memory mapping may be used depending on the application of Processor 100 of the present invention.

In another non-limiting example embodiment 4 of Fig. 1B, Processor 100 may be provided with a PCI-X interface 14 which may be configured to be connected to PCI standard hardware devices 38, 40, 42, as an embodiment. Currently, the PCI-X interface provides a 32 or 64-bit wide transaction oriented bus, at up to 133 MHz. However, faster implementations may be contemplated. Many I/O controller cards and chips are currently available for this wide transaction oriented bus. Cross bridges (XB) 58, 60, 62 may be used to further utilize the PCI-X interface with more hardware devices. The PCI-X interface supports so-called split transactions, i.e., an action may be initiated on one hardware device and, while it is completing, another transaction may be run on a different hardware device.

Commercially available Southbridge chips 44 may be used to interface between the PCI-X interface and the standard hardware devices. Essentially, the PCI-X interface and its controller become invisible to software, which conducts transactions directly with the external device (e.g., Direct Memory Access, DMA).

Processor 100 may also be provided with a synchronous port 16 from which a time division multiplexing (TDM) device may be connected. In one embodiment, a total of 16-pins are provided and configured to support various styles of TDM serial data port. The synchronous port may also be connected to audio codecs (coder/decoder) and transfer continuous streams of data at a strictly defined rate. The synchronous port may also be used for local connections between multiple processors of the present invention.

Processor 100 may be further provided with a standard IEEE test port 18 (e.g., IEEE 1149). In addition to the standard IEEE functions, the port may be configured to support additional commands for loading and examining on-chip registers and memory arrays. The test port may also be used for loading power-up initialization code and program debugging.

Fig. 1C illustrates a clustering of Processor 100 as a multiple processor system 6. The multiple processor system comprises four Processors 100-2, 100-4, 100-6, 100-8 each connected to a Memory Hub (MH) 90 via Memory Buses 20-2, 20-4, 20-6, 20-8. Each of the Processors may have a similar architecture to Processor 100. In an embodiment, each of the Processors may be a video processor with high bandwidth interface. Integrating the video processors with the memory hub may improve the efficiency of the processors benefiting from the direct memory access (DMA).

Each of Memory Buses 20-2, 20-4, 20-6, 20-8 is connected to the memory devices on its own. For example, memories 22-2, 24-2, 26-2, 28-2 are connected to memory bus 20-2 of Processor 100-2. All of the memories is combined into a shared memory space and accessible by all Processors, as required by most multiprocessor operating systems. To this end, the memory buses are enhanced with, for example, a dual-master Bus Request/Grant Mechanism enabling the memory buses to be shared by another chip at the other end of the memory bus from a Processor. Each of the Processors may use the memories on its own bus

directly, or may use the memories on another processor's memory bus via the memory hub using the Bus Request/Grant Mechanism to take over the control of the another bus for the duration of a memory burst access. A Processor (e.g., Processor 100-2) attempting an access to a memory (e.g., Memory 22-4) for another bus (e.g., Bus 20-4) is recognized by the memory hub which is configured to request a temporary mastery of the another bus from its Processor (e.g., Processor 100-4). Processor 100-2 then completes the memory transaction. The memory hub may be configured to arbitrate the processor's competing requests.

Each of the Processors may use atomic lock/unlock instructions acting at the L2 cache level to coordinate the sharing of a global memory area in a way that removes the need to maintain the L1/L2 cache coherency. All of the atomic memory instructions have a feature in common that the variable operated on is in the L2 cache, and is never copied into the L1 cache. With the L1 cache global flushing on unlock and semaphore signal, the L1/L2 cache coherency does not have to be maintained. In a multichip clustered system, only interchip L2 cache coherency need to be maintained, and even then only for the atomic instructions. This has a much lower overhead than a full intercache coherency mechanism. When any Processor executes an atomic memory instruction, this is recognized by the Memory Hub. The memory hub then requests a temporary control of all Processors' memory busses and broadcasts the atomic memory operation to all the Processors, which must maintain L2 cache coherency in this case only. Thus the overheads of inter-L2 cache coherency in a clustered system are incurred only on the atomic memory operations, which are relatively uncommon, while the overwhelming majority of memory operations incur no overhead when accessing a memory directly, and relatively little overhead when accessing a memory indirectly via the memory hub. The cache coherency issue will be described in later sections in more detail.

HARDWARE ARCHITECTURE

Fig. 2 is of a functional block diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing an architecture of Processor 100 of the present invention. Processor 100 comprises Integer Processing Units ("IPUs") 102, 104, 106,

108, a Router 110, a secondary (i.e., level two) cache memory ("L2 cache") 112 and a Mathematical Processing Unit ("MPU") 114. The IPUs are disposed in communication with the L2 cache and the MPU via the Router along with System Buses 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148. In this exemplary architecture, four IPUs 102, 104, 106, 108 are collaborating with single MPU 112 and L2 cache 114 (i.e., a single MPU 112 and L2 cache 114 are shared by four IPUs 102, 104, 106, 108). Alternatively, the number of IPUs, MPUs and L2 caches may vary in other embodiments depending on the requirement of specific applications.

The configuration of Processor 100 will depend on the context of system deployment. Factors such as, but not limited to, the desired throughput of a processor, capacity and/or location of the underlying system, nature of desired system deployment (e.g., portable, desktop, server, and/or the like environments) may affect deployment requirements and configuration. It should be further noted that disparate processing resources may be employed in various quantities. Processing resources may include digital signal processing units (DSPs), graphic processing units (GPUs), IPUs, input/output controller processing units, memory management units (MMUs), MPUs, processing cache memory, vector processing units (VPUs), and/or the like. For example, in certain embodiments such as 3D animation rendering, floating-point throughput is desirable and Processor 100 may be configured with a single IPU and four MPUs interconnected by a Router. In yet another example embodiment, such as for a high traffic web server, handling multiple web requests simultaneously may require Processor 100 configured with 16 IPUs, a single MPU, and larger processing cache interconnected by a Router. Further, in certain environments, processing resources may employ fewer or greater than 32-bit addressing and instruction pathways. For example, for environments where working with large datasets, 64-bit instruction and addressing may be employed for any of the processing resources.

It should be noted that although processing resources on Processor 100 may be external to each other, the architecture and arrangement of the processing resources makes them intra-chip dependent and interactive. This architecture is significantly different from

current and prior art designs. Rather than merely duplicating the functionality of an existing, discrete, monolithic microprocessor design for a system to achieve a multiprocessor system, a radical rethinking and implementation is exhibited by Processor 100 of the present invention. In other words, elemental processing resources (e.g., IPUs, MPUs, L2 caches), which individually may not be capable of servicing an entire instruction set, are dependant on one another. These intra-dependent processing resources, together, service an instruction set more efficiently. Further, through instruction set design and intra-chip networking via a Router, these intra-dependent processing resources are capable of identifying other processing resources to which they may delegate instruction signals that they themselves may be incapable of servicing. This intra-dependent, networked, processing resource, design chip architecture is more efficient than current and prior art monolithic designs. For example, prior art systems may combine two Intel Pentium chips in a system for extra integer performance. However, such a system will be inefficient as such monolithic chip duplication will also duplicate extra mathematical processing logic, memory management logic, etc., which may go underutilized. Furthermore, such prior art designs depend on processor intensive and time expensive operating system scheduling to make use of the added processing facilities. Processor 100 of the present invention does not suffer from these limitations. It may employ numerous IPUs to enhance integer processing throughput, and itself may delegate instruction processing to the proper processing resources without duplicating underutilized portions of a chip.

IPUs 102, 104, 106, 108 are the general integer and logic processing units of Processor 100 and each of the IPUs is responsible for running a separate program thread (i.e., a part of a program that can be executed independently of other IPUs). Furthermore, each of the IPUs may run independent processes and program software. In one embodiment, each IPU may be a simple 32 bit microcomputer with 8x32 bit data registers (D0-7), 8x32 bit address registers (A0-7), a 32-bit instruction pointer (IP) and a 32 bit return link pointer (RP). Address register A7 may be used as a stack pointer. Alternatively, each IPU may be a 32-bit or 64-bit microcomputer. A 32-bit IPU may have a bank of 16 general purpose registers of

32 bits, organized as 8 address registers and 8 data registers. In a 32-bit implementation of Processor 100, 64 bit operations may take an extra cycle (i.e., double precision operations). A 64-bit IPU may have a bank of 16 general purpose registers of 64-bits, organized as 8 address registers and 8 data registers.

Each of IPUs 102, 104, 106, 108 is configured to execute the instructions of a thread and/or process and perform relatively simple calculations of the instructions such as add, subtract, basic logic (e.g., Boolean operations) and/or integer calculations. Each of the IPUs are further configured, upon decoding an instruction while executing the instructions of the thread and/or process, to send calculation requests along with data and opcode of the instruction to MPU 114 for relatively complex calculations such as multiplication, division and/or floating point calculations. Alternatively, the IPUs may be configured to send the instructions themselves that require complex calculations to the MPU without decoding the instructions. Similarly, each of the IPUs may also be configured, while executing the instructions of the thread and/or process, to send access requests to L2 cache 112 for accessing data and/or instructions stored in the L2 cache.

Each of IPUs 102, 104, 106, 108 includes at least one level of internal cache ("L1 cache") such as L1 caches 103, 105, 107, 109 and each of the L1 caches of the IPUs may be subdivided into areas for instruction cache (I-cache) and data cache (D-cache). The D-cache is further subdivided into local and global areas using a hash function to increase the cache hit rates as compared to the uniform cache of a similar size. The hash function is described in more detail in Fig. 5.

Router 110 is an instruction signal switching hub between the processing resources such as the IPUs, L2 cache and MPU. In one non-limiting example embodiment, the router may employ a cross-point switch. The router receives requests from the IPUs via system buses 116, 120, 124, 128 and routes them to target resources (e.g., L2 cache 112 or MPU 114) via system buses 132, 136, 140. The request competition between the IPUs may be resolved with different level of priorities (e.g., low, medium and high) assigned to the IPUs at a given time adding the flexibility of the access priority. Fig. 12 describes the access

priority in more detail. In an alternative embodiment, the router may arbitrate the requests from the IPUs based on a round robin priority method. The router may be configured to send acknowledgements to the IPUs via response paths 118, 122, 126, 130 in response to the requests from the IPUs. The router may also be configured to receive acknowledgements from the L2 cache and MPU via return paths 134, 138, 142.

L2 cache 112 is an external cache to IPUs 102, 104, 106, 108 and shared by the IPUs. The L2 cache receives access requests from the IPUs via router 110 along with instruction bus 132 and data bus 136, respectively. The requested data and instructions are delivered to the requesting IPUs via data bus 144 and instruction bus 146, respectively. Similar to the L1 caches, the shared L2 cache is also subdivided into areas for instruction cache (I-cache) and data cache (D-cache), and the D-cache is further subdivided into local and global areas to increase the cache hit rate. In an alternative embodiment, the L2 cache is unified for instructions and global data, while still having a separate local data area to increase the hit rate. Additionally, the instruction and data busses may be combined, as instruction/data busses, for better bus utilization. The L2 cache includes ports 150, 152, 154 so that the peripheral devices such as memory, PCI-X and TDM devices may be connected to the L2 cache through the ports.

L2 cache 112 may be configured with a built-in arithmetic logic unit to perform several control operations directed to the L2 cache management and memory control as well. The control operations of the L2 cache include the execution of a small set of atomic instructions, lock/unlock functions, read-modify-write functions and instruction/data delivery functions. The L2 cache may be lightly pipelined to accept multiple requests from other IPUs before the completion of a request from an IPU (i.e., accept new requests on each cycle from the IPUs while others are in progress). Further, the L2 cache may be optimized by having some directly addressed, non L2 cached memory at the L2 cache level of the chip, accessed at L2 cache speed. Critical interrupts and operating system code can be kept here, equivalent to a portion of the address space that is always cached. More detailed descriptions regarding the control operations in the L2 cache will follow in later sections.

MPU 114, one of the other shared resources in Processor 100, is a complement processing resource for IPUs 102, 104, 106, 108 and is configured to perform relatively complex calculations of instructions such as multiply, divide and/or floating point calculations upon receiving requests from the IPUs. The MPU is configured to receive calculation requests from the IPUs with decoded opcodes and data via Router 110 along with data bus 140. Alternatively, the MPU may be configured to receive and execute encoded instructions with complex calculations forwarded from the IPUs. The calculation results are delivered via return data bus 148 to the requesting IPUs. The MPU may be lightly pipelined to deal with multiple requests from the IPUs before the completion of a request from an IPU (i.e., accepts new requests from the IPUs while others are in progress). For example, the MPU is pipelined with separate short pipes for floating point add/subtract, and integer/floating point multiply/divide operations. A state machine implements multi-sequence iterative functions such as integer/floating point divide, square root, polynomials and sum of products. These in turn may be used to efficiently implement more complex functions.

In one non-limiting implementation of Processor 100, a single MPU (e.g., MPU 114) is shared by multiple IPUs (e.g., IPUs 102, 104, 106, 108) using the silicon area more effectively when compared to a conventional processor with a similar size. The streamlined architecture of the IPU structure (e.g., 32 bit microcomputer) makes it practical to incorporate several of the IPUs on a single chip. Current conventional designs become even more inefficient in contrast to Processor 100 of the present invention as the effective area of silicon required becomes even greater when such conventional processors are duplicated and employed as monolithic structures in parallel.

The clock speed of Processor 100 may be modest when compared to the conventional processor and yet achieve similar processing throughput because greater execution throughput may be achieved by multiple processing resources (e.g., IPUs) running in concert without requiring higher clock rates. For example, Processor 100, a design with an MPU shared by four IPUs, may use a clock speed of 533 MHz for the same amount of, or even

higher throughput compared to a conventional processor running at 2 GHz. The reduced clock speed renders several benefits including less complicated circuitry, less stages of pipelines, less heat problems and, reduced cost with improved accuracy and capabilities. Furthermore, such a design of the present invention allows for modern processing technologies to be employed in environments too harsh to employ conventional processor technology. For example, certain aerospace environments must employ old Intel 80386 processor technology because the harshness of space requires relatively large die cast mask implementations (e.g., 35 nm) as contrasted with current state of the art fabrication implementations (e.g., 10 nm), which reduces the amount of logic that may be supported on a single die. Processor 100 of the present invention may be implemented employing such a course fabrication technology utilizing the available silicon area more efficiently even at relatively low clock rates.

Fig. 3 is of a simplified schematic diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of the data paths of IPU requests from IPUs 102, 104, 106, 108 to a target processing resource such as MPU 114 of Processor 100.

If the arithmetic operation required is more complex than integer add, subtract or compare, or floating point sign/magnitude compare, the operation may not be executed by the IPU. Instead the B and C parameters of the complex arithmetic operation are loaded into the operand pipeline registers and a request issued for access to the shared MPU, which can accept a new operation into its pipeline or pipelines every clock cycle. If more than one IPUs request MPU service in the same cycle, the request from the IPU which is running at the highest priority is granted. The requests from the IPUs are pipelined with pipeline input registers. A decoder decodes the requests determining the requesting IPU number. The decoded number of a requesting IPU may be added to the output result at the pipeline output register. In the execute pipeline phase, an IPU may issue a request to the Router to connect to the shared MPU to execute, for example, a complex math instruction. On the next free cycle, the IPU may write the function parameters to the input registers of the MPU. If the instruction requires double precision, the writing of the function parameters may take two

cycles. When the MPU has calculated the result, it returns the result on the dedicated return bus to all IPUs, with the IPU number of the corresponding request so the correct IPU will accept it. If the result is double precision, this may take two cycles as well.

Because a requesting IPU sleeps waiting for the result from the MPU, adjacent operations in the pipeline cannot be for the same IPU. This simplifies register bypassing in the MPU, eliminating the need for much of it.

The command set for the MPU operations includes integer multiply and divide, plus a full set of floating point operations. There is also a function pipeline which supports the usual set of logarithmic and trigonometric functions. There are enough spare operation commands to allow for further specialized coprocessors operating on similar principles.

Fig. 4 is of a simplified schematic diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of the data path of an IPU such as IPUs 102, 104, 106, 108 of Processor 100. An exemplary embodiment of the IPU employs a 32-bit data bus structure as shown in Fig. 4. The data bus provides a data path for the various components comprising the IPU. Instructions are fetched from the memory devices (e.g., L-1 cache, L-2 cache or external memory devices) and decoded in an instruction decoder as shown in Fig. 6.

Every enabled clock cycle, the data path pipeline executes a new instruction. In the first data path pipeline stage, either a pair of registers or a register and a literal constant are loaded into two pipeline operand data registers, B and C, controlled by only a little instruction predecode logic during the instruction fetch pipeline stage. During this stage, the instruction is more fully decoded for the following stage. These registers include 16 general purpose registers (e.g., 16x32 bit registers) and the various other registers such as the instruction pointer or status register. Subsequently, the decoded addresses and data are stored at a 16x32 bit register space. The register space provides a set of high speed memory stores for the IPU and has three inputs for the selection of one of the 16 registers. The one more required bits for the selection of the registers come from the instruction decoder that interprets the operand of an instruction. Figs. 21 and 22 describe this aspect of the invention in more detail.

The register space is connected to a general purpose Arithmetic Logic Unit (ALU) via multiplexers and the 32-bit operand pipeline registers. The ALU provides the arithmetic computations for data stored in the register space. In the following data path pipeline stage, the decoded operands are passed either to the ALU or to MPU 114 depending on the type of the required calculation. Multiplexers have an input from a status register, which stores selection information among the general purpose registers. The status register further stores information such as instruction pointers, return pointers, and/or the like. The ALU is configured to simultaneously execute an arithmetic add or subtract and a logic operation such as move, and not, or, xor, prioretize, barrel shift or rotate. Comparison operations include signed, unsigned, float and double. Minimum and maximum operations, signed, unsigned, float and double, are executed by doing a comparison operation of B and C in the arithmetic unit and using the result to select whether the logic unit passes B or C. Double precision operations are implemented as two single precision operations, with appropriate handling of zero and carry flags. Bit test and set operators return the zero flag according to the bit value at the test, before the set.

The output of the ALU is connected to the caches such as L1 and L2 through a multiplexer and register. The outputs from both the adder and the logic operations are available simultaneously, and pass through a number of multiplexors to a number of pipeline registers, each of which may independently be loaded with either of these outputs, or with the result of the MPU or memory access where appropriate. The register writeback pipeline register can be loaded with any of these, and it or they can be bypass multiplexed into the B or C operand registers if either one is the result register of the previous instruction or the one before that.

Alternatively, either the B or C register can be loaded from the result that is not being written back to a register. This is used to roughly halve the cycles in multicycle instructions by supporting two simultaneous data flows. This allows for interruptible multicycle vector operations to execute at about twice the speed of an equivalent software loop. The set of

vector operations is extensive, covering most of the C string library plus vectored arithmetic including inner product.

The ALU may shift the C operand by 0-7 bits; left, right or rotate. This is used to scale subscripts for bit, byte, short, integer or long, or for part barrel shift. In the logic return data path, combined with both the read data and write data, there is a byte rotator which either completes a barrel shift or rotates memory data to alignment, and a byte, short or integer zero or sign extender. Memory write data also passed through these. Whichever source is multiplexed to the register writeback pipeline register, to be written to the actual register in the following cycle, has its set of flags selected to be stored in the status register.

The memory data address register may be loaded from either the arithmetic or logical units, allowing for all address arithmetic to take only one cycle, even autoincrement or decrement. The following cycle the data from the L1 D-cache is read unless there is a cache miss, and the cycle after that the address can be written to.

When the address register is loaded, a hash function of the address is computed and also loaded into a register. This hash is used to select a row in the cache in which the data and address are stored. Additional multiplexers and registers are configured to perform the hash function incorporating data type categorization requirements allowing locality information determination of the output data, thereby the output data may be apportioned and stored correctly between the local and global areas of the caches. When address arithmetic is performed, a hashed address may also be computed for L1 cache access. In the case of a data address, the highest bit of the hashed address may be a local/global classification, based on a bit-masked comparison of the data address and the local stack segment address. This may be varied to allow either local or global allocations to occupy greater or lesser percentages of cache space. The memory address hash function is special in that it distinguishes between local and global data in order to favor the former. The lower bits of the address register and the number of bytes being read or written are combined to emit byte strobes to the L1 D-cache, which are used to gate individual bytes and to control nonaligned split word reads or

writes, which require extra cycles. Thus read or writes do not have to be aligned, although they use extra cycles if they are not. This also minimizes the miss rate of the L1 D-cache.

Each of Branched Data Buses are connected to Router 110 for transmitting a request to the shared resources (e.g., L2-cache 112 and MPU 114). For example, the operand and/or data of an instruction that requires a complex calculation may be sent to the shared resources (e.g., MPU 114) before the ALU of the IPU performs the calculation.

Fig. 5 is of a logic diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of the hashing function which is connected to each of the outputs of IPUs 102, 104, 106, 108.

A cache memory (e.g., L1 cache or L2 cache) is addressed by a hashing function of the data address, which selects a cache memory row that the data may be stored in. This hashing function is usually the exclusive or (XOR) function of some of the address bits to a smaller number of bits. In this invention, when a data address is hashed, most bits are generated by a simple XOR hashing function, but one or more bits are used to subdivide the data cache into a local zone and a global zone. It is known that local data has much higher reuse rate than global data and it is desirable to bias the cache memory towards storing local data. As instructions cannot execute from local memory, instruction addresses are assumed global.

In an exemplary embodiment of a hashing function of the present invention, the top bit of the hashed address divides the data cache (e.g., L1 cache or L2 cache) into two halves. This bit is generated by determining if an address is local, that is that the address is within the limits of a segment of memory allocated by the operating system for use as the local data stack of the current process. This test may be made by simple logical comparisons of the address against a stack segment address register and a stack segment bitmask register. These segments are set up by the operating system when it allocates local memory for a process, and are not modifiable by a user program. The Stack Mask (SM) register is a bitmask that selects the high bits of the stack pointer address, which defines the size and limits of the stack segment. If the masked high bits of an address are equal to the masked high bits of the Stack

Segment (SS) register, the address may be determined as a local. The low bits of the Stack Segment register define the size and limits of a system reserved area at the base of the stack segment. If an address is within this limit, it may be determined as local but not user usable, resulting in an interrupt to the operating system if in user mode, detecting stack overflow before it can damage other processes. Interrupts or operating system calls can use this area safely. If an operation alters the Stack Pointer register to a value not in the limits of the stack segment this also results in a stack overflow interrupt. Thus a user program cannot overflow its stack without being stopped before it can cause damage. This division of the data cache into local and global areas extends not only to the L1 cache but also to the L2 cache, because local data are not shared between processes.

Fig. 6 is of a simplified schematic diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of an instruction decoder of an IPU such as IPUs 102, 104, 106, 108 of Processor 100. IPU instructions fetched from L1/L2 caches or memory devices are decoded in the instruction decoder and used in the IPUs for further execution. In an embodiment of the present invention, all IPU instructions are 16 bits in size. The IPU instructions are aligned in 32 or 64 bit words depending on whether the instructions are executed in a 32 bit IPU or a 64 bit IPU. Both the 32 bit and 64 bit IPUs may have the same architecture as shown in Fig. 6, but differ in word size and detail of instruction set.

IPU Instructions are executed in a pipeline with a single or multicycle operation. Some of IPU instructions may contain a small literal constant, which may be extended with one or more literal prefix instructions. Literal prefixes are saved in the literal register until the prefixed instruction are executed in the IPU. Literal prefix instructions in the same word as the prefixed instruction are executed simultaneously, and thus, more than one instructions may be executed in a cycle (i.e., superscalar execution). The presence or absence of literal prefixes before an instruction constitutes a 17^{th} instruction bit, which increases the number of instructions that can be encoded in 16 bits. There are some instructions that are always prefixed and some instructions that are never prefixed. These instructions share 16 bit

opcodes and are distinguished by the 17th bit. The literal prefix instructions are described in more detail with Fig18.

While executing an instruction, the next instruction is fetched from the L1 or L2 I-cache or from external memory and predecoded. When the current instruction has executed, the next instruction is loaded into the instruction register and decoded in parallel with register fetch, which is controlled by the small amount of predecode logic. A literal constant, with any prefixes, is made available simultaneously with register fetch. In the following pipeline stage, the arithmetic and logic unit (ALU) executes the decoded instruction. The last pipeline stage is register writeback.

All controls for the data path pipeline come from the instruction decoder. Feedback from the instruction decoder controls multicycle instructions by modifying the instruction register and reexecuting it. Thus to the data path pipeline, a complex multicycle instruction appears to be a sequence of simple single cycle instructions. Even interruptible multicycle vector instructions are controlled in this simple manner. When a jump is taken, the following instruction has been fetched and is ready to be loaded into the instruction register, while the instruction jumped to is not fetched until the next cycle. The following instruction is annulled by loading it as a no operation and otherwise executing it normally. This is a pipeline bubble, and consumes roughly the same power as any other operation.

Instruction execution can sleep if there is an L1 cache miss or an MPU operation. In such a sleep, the pipeline clock is disabled and very little power is consumed until the memory or MPU operation completes and the clock is reenabled. The atomic lock operations also cause sleep, until the unlock operation. The sleep lock/unlock operations will be described in more detail with Figs. 26, 27.

The pipeline has been designed to be as short, but the architecture is compatible with a longer pipeline if desired for higher clock frequency. It is a tradeoff between clock speed and the number of cycles per instruction.

Fig. 7 is of a simplified schematic diagram illustrating an alternative, non-limiting example, inventive aspect of the present disclosure of the data path of an IPU such as IPUs

102, 104, 106, 108 of Processor 100. The IPU is presented incorporating a 64-bit data bus structure. A 64-bit Arithmetic Logic Unit (ALU) is connected to a 64-bit general purpose register via multiplexers and 64-bit registers. The general flow of data of the IPU in this alternative embodiment is similar to the IPU of Fig. 4 with a 32-bit ALU.

Fig. 8 is of a simplified schematic diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of the instruction/data path of L2 cache 112 of Processor 100. If there is a miss on accessing the L1 I or D-cache, the IPU pipeline sleeps and a request is made for access to the shared L2 cache. Atomic memory operations bypass the L1 D-cache and directly request L2 cache access. The L2 cache access normally requires several clock cycles and is pipelined enabling the L2 cache to accept a new request every clock cycle.

The L2 cache has its own extra circuitry (e.g., a dedicated ALU) which allow the L2 cache to implement a set of atomic memory operations including a form of global atomic lock/unlock which is much more efficient than the conventional spinlocks. The extra circuitry may supports atomic read/modify/write commands in the L2 cache. With the read-modify-write operations in the L2 cache, the write data from the IPU is not written to the L2 cache as usual. Rather, the write data is combined with the previous data at the address and then written. The combination is either a simple arithmetic add or one of a few logical bit mask operations. The read-modify-write instructions are executed in the L2 cache using its own dedicated ALU multiplexed with its input and output pipeline registers. The L2 data cache input is configured to block further requests while executing such instructions, making them atomic instructions between the IPUs. This enables synchronization between the IPUs, obsoleting the spinlocks of the conventional processor designs. In the atomic instructions, the data at the address is first read into the L2 data cache output register, followed immediately by a write of data to the same address. Then, writing to the L2 cache ALU with the data read from the address allows for the result to be written to the address, atomically. Some of the possible L2 cache ALU operations include add, subtract, bit set and bit clear, which allow for atomic implementations of lock waiting, event waiting and semaphores.

When an IPU executes a lock instruction, it sleeps if it cannot immediately acquire the lock. When an IPU executes an unlock instruction, the L2 data cache sends the result on the data bus tagged as unlock, with the IPU address. All IPUs that are asleep and/or awaiting an unlock of that address will wake and retry their lock instructions. If one succeeds and resumes execution, any others may continue to sleep until another unlock instruction frees the resource they require. Similar instructions allow IPUs to sleep awaiting an event bit to be set in an event bit mask. Other instructions implement semaphores, conditionally trapping to the operating system if a semaphore requires run or wait queue manipulation.

The L2 cache is not required to maintain cache coherency with each IPU's L1 cache. This is because disciplined programming is required to share global data, using global atomic lock, unlock or semaphore instructions to control access. A semaphore instruction is simply an atomic increment or decrement on the lock variable, with a software interrupt generated on the basis of the flags generated by the atomic operation, which invokes the operating system scheduler if needed. On an unlock or semaphore signal of departure from a critical region of code in which access to shared data is regulated, the L1 D-cache is flushed to L2 in order to purge the L1 cache of the very data it had regulated access to. This ensures cache coherency when needed, without any performance penalty otherwise. Cache coherency need only be maintained between the L2 caches of multiple chips in a cluster as shown in Fig. 1C, and even then only for atomic operations, which are few enough that they can afford an extra cycle for coherency. The L2 cache may also be flushed to off chip memory under program control as needed in device drivers.

GENERAL OPERATION

The general operation of Processor 100 will now be described with reference to the exemplary data processing system and architecture of Figs. 1A, 1B and Fig. 2.

Operating systems that support multi-threading operation of application programs enable programmers to design the programs whose threaded parts can be executed concurrently. Further, modern operating systems generally run a substantial number of individual program processes at any given time. Accordingly, each of IPUs 102, 104, 106,

108 may run a separate program process and/or thread (hereinafter thread and process may be used interchangeably, unless otherwise specified) allocated by an operating system independently. Many operating systems schedule the instructions of a thread/process and thereby instructions and data for each thread allocated to an IPU is fetched and stored in the L1 cache of the IPU as well as in L2 cache 112 in advance. Alternatively, the IPU may fetch the instructions and data from the external memories into its relevant registers. Subsequently, the executed instructions and data may then be stored into the L1 cache as well as the L2 cache.

Each of IPUs 102, 104, 106, 108 executes instructions of an allocated thread by following processor fetch-execute-store sequences. Each of the IPUs may access their local caches (i.e., L1 caches 103, 105, 107, 109) to obtain information (i.e., instructions and/or data) required to execute the instructions. Each of the IPUs may also access shared L2 cache 112 through Router 110 to obtain the necessary information. The requested information from an IPU may then be delivered by the L2 cache via dedicated return paths 144, 146. If the required information is not found in the L2 cache, external memories 22, 24, 26, 28 may also be accessed by the L2 cache.

While executing allocated instructions from an execution thread in an IPU (e.g., IPU 102), an instruction that requires complicated calculation such as floating point and/or vector math calculation is detected during the decoding process of the instruction. Subsequently, IPU 102 extracts the operand (e.g., floating point number etc.) and opcode (e.g., multiply, divide etc.) from the instruction and sends them to MPU 114 through System Bus 116 and Router 110. Alternatively, the instruction with complicated calculation itself may be sent to the MPU without extracting the operand and opcode.

MPU 114, upon receiving a request from IPU 102 (i.e., a requesting IPU), performs the requested calculations based on the operand and opcode employing a fetch-execute-store sequence. The calculation results may then be delivered via dedicated return path 148 to the requesting IPU (i.e., IPU 102).

While executing allocated instructions of a thread in an IPU (e.g., IPU 102), during decoding, the IPU may detect an instruction that requires information from a region in memory that happens also to be in L2 cache 112. Subsequently, IPU 102 extracts from the instruction an address for the L2 cache and sends an access request to the L2 cache directed at the address through system bus 116 and Router 110. The L2 cache 112, upon receiving the request from IPU 102, accesses to the requested address and provides information (e.g., address or data) stored at the address to IPU 102 through dedicated return paths (i.e., buses 144, 146) by placing the information on the buses.

In particular, after a requesting IPU (e.g., IPU 102) provides a request to another processing resource (e.g., L2 cache 112 and/or MPU 114) independently, the requesting IPU may go to "sleep" until a responding processing resource (e.g., L2 cache 112 and/or MPU 114) returns the requested information. Sleep is an important capability of Processor 100. Sleep may be induced by a switch unit in an embodiment, which may be placed between a power or clock providing line-in to any processing resource. Upon instruction, the switch unit may act to switch off power or clock to the unit while waiting for a response thereby reducing the overall power requirements for Processor 100. The resulting outputs from a sleeping processing resource may be configured to be non-existing, and as such they do not interfere with other competing inputs. For example, if IPU 102 or MPU 114 is put to sleep, their outputs 116, 142 and 148 may not affect and are no longer interpreted as inputs at their destinations. The switch unit may be switched on by other processing units, typically, upon completion of a request at a shared resource (e.g., MPU 114) and/or freeing of a processing resource (e.g., a locked memory location being unlocked in L2 cache 112). Alternatively, various processing resources may be shut off upon instructional direction so as to save power. For example, in an environment where conserving power is desirable (e.g., a battery powered laptop computer), a program (e.g., an operating system) may instruct the processor to shut off various processing resources to extend battery longevity. In yet another alternative embodiment, such a program may dynamically turn on and off various processing resources to maintain a desired level of power draw while maximizing processing

throughput. In yet another embodiment, Processor 100 itself shuts off processing resources while idling. Alternatively, the delegating processing resource (e.g., IPUs 102, 104, 106, 108) may continue to operate when requested for concurrent processes.

The requesting processing resource (e.g., IPU 102) may "sleep" even longer if the servicing processing resources (e.g., L2 cache 112 and/or the MPU 114) are currently busy processing requests from other requesting resources. For example, after IPU 102 requests information (i.e., instruction and/or data) from L2 cache 112 with an address via Router 110, IPU 102 goes to "sleep" until the requested information returns from the L2 cache via the return paths (i.e., system bus 144, 146). The "sleeping" time duration for IPU 102 after the request depends on the status of the L2 cache as well as the priority level of requesting IPU 102. If the L2 cache can return the requested information to IPU 102 right away, IPU 102 may resume the next operation within a relatively short time period. However, if the address of the L2 cache is locked by another IPU (e.g., another IPU is using the area for the execution of semaphore instructions), IPU 102 has to "sleep" longer until the lock is released by the current owner of the address. The L2 cache may be configured to inform the requesting IPU (e.g., IPU 102) when a locked area of the L2 cache is released. That is, when the current owner releases a locked area, the L2 cache broadcasts the lock release information with an identification of a "sleeping" resource (e.g., IPU 102) via dedicated return path (e.g., system buses 144, 146). Upon receiving the lock release information from the L2 cache with the sleeping identifier, the "sleeping" resource "wakes up" and may retry its access request to the L2 cache. Alternatively, the L2 cache may return the requested information (e.g., address, data and/or instructions), not the lock release information, to the requesting IPU and thereby let the IPU resume processing using the received information from the L2 cache.

The collaboration between IPUs 102, 104, 106, 108 and MPU 114 uses a similar "sleep" and "wake up" protocol as described above with regard to the collaboration between the IPUs and L2 cache 112. For example, after a requesting IPU (e.g., IPU 102) requests a calculation from the MPU along with information (e.g., data and opcode) via Router 110, the

requesting IPU goes to "sleep" until the requested information returns from the MPU via the return paths (e.g., system bus 148). The "sleeping" time duration for the requesting IPU after the request may vary depending on the status of the MPU as well as the priority level of the requesting IPU. If the MPU can return the result of the requested information to the requesting IPU right away, the requesting IPU may resume the next operation within a relatively short time period. However, if the MPU is busy with requests from other IPUs, the requesting IPU has to "sleep" longer until the MPU returns the result. Similar to the L2 cache, the MPU is also configured to broadcast the calculation result with an identification of the requesting IPU (i.e., IPU 102) to all of the IPUs via return path 148. Upon receiving the calculation result from the MPU with the identification, the "sleeping" IPU (i.e., IPU 102) "wakes up" to resume process using the returned result.

In every clock cycle, every IPU may send a request to one of the shared resources (i.e., L2 cache 112 and MPU 114). Since each shared resource can accept only one request per cycle, requests from the IPUs may be accepted according to a dynamic priority system. The other requests from non-selected requests may be deferred to next cycles and will be accepted later, one per cycle. For example, each IPU may be assigned a low (0), medium (1) or high (2) priority at any given time. The low priority may indicate that the IPU is running a program normally. The medium priority may indicate that the IPU is running with a lock or servicing an interrupt temporarily. The high priority may indicate that the IPU is running with a super-priority instructions set (e.g., running no program and servicing interrupts only). When this priority is not enough to select an IPU because of multiple requesters at the same time, another priority scheme such as the round robin priority may be used based on the last satisfied request. The priority determination hardware may contact the multiplexers of Router 110 that routes the selected request to the input pipeline registers of each of the shared resources along with the requesting IPU number, request code, request operands, etc.

Several cycles later, the result from the shared resources may be broadcast on shared result buses 144, 146, 148 to all IPUs. Each sleeping switch unit at an IPU "looks" at the result buses for its IPU number. When a sleeping IPU detects a match, the IPU routes the

broadcast result to its register and wakes up from the sleep. Sleeping may be accomplished by gating the clock off to the requesting IPU with a switch unit, and waking by gating the clock on again. However, the lock/unlock request, which is a request to the L2 D-cache, may be configured as an exception. For example, if the lock request is not granted on the first cycle, it is simply withdrawn and retried only after an unlock on the same address. The result of the lock/unlock request may simply be the address of the lock. Not only does this address signal from the L2 cache wake up the IPU that requested the unlock, but it also wakes any IPU sleeping on the same lock address, causing them to retry their lock request. Alternatively, the L2 cache may be configured to return the requested results from the requesting IPUs depending on the priority levels of the IPUs.

Additionally, a uniquely designed cache policy between L1 caches 103, 105, 107 109 and L2 cache 112 of the present invention makes it possible to manage the caches without using a complicated protocol as is used in conventional multiprocessor architectures. For example, after an IPU (e.g., IPU 102) executes semaphore instructions completing a high level critical region, or after IPU 102 executes lock/unlock instructions completing a low level critical region, all the hot global data in the corresponding L1 cache 103 is flushed and copied to the L2 cache global area. The hot global data indicates accessed and modified data during the execution of, for example, the semaphore instructions and/or lock/unlock instructions. Other data except the hot global data may be simply discarded. Further, when IPU 102 reschedules threads, unless the next thread is the same as before, the local and global data in L1 cache 103 is flushed to the local and global areas of the L2 cache. These policies substantially eliminate cache coherency problems between the L1 caches and the L2 cache.

Although, the general operation of the processor was described with regard to IPU 102, other IPUs 104, 106, 108, independently and concurrently operated, have similar capabilities and the same explanation may be equally applied to the other IPUs.

REQUEST DELEGATION

Fig. 9 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a resource delegation process in IPUs 102, 104, 106, 108 of Processor 100 in which a requesting IPU constructs and delegates a request to other processing resources (e.g., L2 cache 112 and/or MPU 114).

The general operation iterative flow for a delegation process in one of an IPU (e.g., IPU 102) may be summarized as: (1) the constructing of a request if necessary to a delegated resource, (2) routing the request, and (3) obtaining a response from the delegated resource at the requesting unit. At some point, an instruction is fetched for the requesting IPU 903. Upon obtaining an execution-instruction, a determination is made by the IPU if the instruction is intended for execution on one of the other processing resources (e.g., L2 cache 112 and/or MPU 114) based on the decoding of the opcode of the instruction 905. For example, if the opcode indicates that an adding or subtracting calculation is required, and no information is required from the external cache (i.e., L2 cache 112), the instruction then may be executed in the IPU without delegation 907. If, however, the opcode indicates that at least one of the other processing resources is required to service the instruction (e.g., a multiplication or division calculations, or information from the L2 cache is required for the execution), then a request may be constructed by the IPU 909. For example, in constructing the request, the IPU may combine several information such as the IPU number, decoded opcode, values from the status registers and priority bit status registers (e.g., lock/unlock/interrupt hardware priority bit). Subsequently, the IPU provides the constructed request to Router 110 911. Upon providing the request to the Router, the IPU is put to sleep and waits for a reply or for a request for the IPU to wake 913.

Fig. 10 is of a block diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a form of a delegation request constructed at a requesting IPU and directed to MPU 114. The exemplary MPU request includes information regarding the ID number of an IPU (1001), opcode of an instruction (1003), data (1005), and priority level (1007).

The IPU ID number component (1001) comprises enough bits to uniquely identify an IPU. For example, with a two-bit IPU number, an IPU may designate any one of IPUs 103, 105, 107, 109 and provides an identifier so that subsequent receiving and delegated resources may properly address a reply. In one non-limiting example embodiment, such an IPU number is provided in hardwired form within the IPU, but in alternative embodiments, such a value may be provided within a specialized register. Of course, a single IPU embodiment may be employed where no IPU number is required.

The exemplary MPU request includes a ten-bit operation code component (1003). The ten-bit opcode section defines the calculation to be performed in the MPU (e.g., multiply or divide etc). The opcode section may also act as an identifier for a destination processing resource (i.e., MPU 114). This pointer may be provided by decoding the operation code, which inherently targets a resource capable of processing the operation. The operation-code may be obtained from the opcode part of the instruction register. For example, if the opcode indicates that a multiplication or dividing calculation is required, it can be determined that the destination resource is the MPU. If, however, the opcode indicates that a read operation is required, it can be determined that the destination resource is L2 cache 112, not the MPU.

The exemplary MPU request may also include a data component (1005). The data component comprises 64-bits and may also include an operand. The values for the data section may be obtained from data registers within a requesting IPU.

The exemplary MPU request may also include a priority component (1007). The priority component may comprise 2 bits and the values may be obtained from the status register of a requesting IPU. The priority bit may represent a lock, an unlock, a hardware interrupt and/or the like, as provided either by the requesting IPU or by software generated interrupts.

Fig. 11 is of a block diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a form of a delegation request constructed at a requesting IPU and directed to L2 cache 112. Similar to the MPU request, the exemplary L2 cache request includes information regarding the ID number of an IPU 1101, opcode of an

instruction 1103, data 1105, address 1107 and priority level 1109. In particular, the L2 cache request may include a 32-bit address component 1107 which designates the targeted address of the L2 cache and/or memory.

Fig. 12 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of request handling in Router 110 of Processor 100 routing requests between the requesting processing resources and servicing processing resources. It should be noted that throughout, for sake of clarity, IPUs 102, 104, 106, 108 may be illustrative of any requesting processing resources, while L2 cache 112 and/or MPU 114 may be illustrative of any servicing processing resources. Regardless of the types of the requests (e.g., an MPU request or a cache request), each request is constructed within a requesting processing resource and provided to Router 110 as a unitary request.

A requesting IPU (e.g., IPU 102) delegates a request to Router 110 that further routes the request to a resource capable of processing the delegated request 1203. As the request is provided, the router iterates a priority deadlock bit 1205. The priority deadlock bit in an exemplary embodiment is within the Router itself, but in alternative embodiments the priority deadlock bit may be located elsewhere within Processor 100. By iterating the deadlock bit, the Router is capable of properly managing and forwarding the request to a resource that is available and prevents the forwarding of a request to resources that are busy. Further, the deadlock bit resolves the situation when more than one requesting entity is requesting the same resource. Depending upon the number and types of shared resources available within Processor 100, the number of deadlock bits may be expanded or reduced so that proper routing to each of the available resources within Processor 100 may be achieved. Thus, with $\log_2$ (the number of requesting IPUs) bits, the deadlock bits will ensure proper routing. For example, for four requesting IPUs, two deadlock bits ($\log_2 4$) are required.

At some point, Router 110 examines the priority bits of a request 1207 based on the information from the priority deadlock bit. Subsequently, by examining the ID numbers of the requesting IPU and information from the priority bits of the request, the Router determines the priority of the request and also determines the identity of a servicing

processing resource (e.g., L2 cache 112 or MPU 114) 1209. Thus, upon being presented with a request, the Router will make a deterministic decision to which resource a request ought to be provided.

Router 110 then determines whether more than one IPU is requesting the same resource 1211. If only one IPU requested the same resource, then the Router may simply select the request 1213. Further, the Router determines whether more than one IPU has presented the Router with a request at the same priority (e.g., a priority at the highest setting) 1215. If it is determined that there are no requests with the same priority, the Router may simply select one request with a highest priority 1213. If it is determined that there is more than one IPU with the same priority, the Router may select a presented request based on the deadlock bit 1215. In one non-limiting example embodiment, the deadlock between two or more requests with the same priority is broken by selecting the request that is closest to the iterated identifier of the deadlock bit. For example, if there are four IPUs making the requests, the deadlock bits may be iterated from 0 to 3, where each state of iteration represents each of the four IPUs that potentially may make a request. In such an example, if the deadlock bit has been iterated to a value of one and the two requests that have the same priority are emanating from IPUs number 3 and 4, then the request coming from IPU number 3 will win the deadlock because its value (e.g., IPU number 3) is closer to the current deadlock state (e.g., deadlock counter value 1).

Upon selecting a request 1213, 1215, the request chosen is forwarded to the input register of the targeted resource minus any priority bits 1217. Alternatively, priority bits are preserved so that there are multiple levels of delegation that may be achieved. Also, a request-granted-acknowledgement signal may be sent to the requesting IPU 1219. This may have the effect of clearing the IPU opcode in requesting IPU that sent the request 1221. Processing then may continue and iterate at the Router 1223.

Fig. 13 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of a response from a target processing resource (e.g., L2 cache 112 or MPU 114) to the requesting processing resource (e.g., IPU 102, 104, 106, 108). A request

from an IPU is received into registers of a desired processing resource (i.e., L2 cache 112 or MPU 114) via Router 110 1303. Upon receiving such a request, the desired processing resource executes the request 1305. Such execution may involve accessing an address at the L2 cache or performing a calculation at the MPU based on the opcode provided from the request. Upon processing the request, a response is prepared into a result register of the target processing resource, which includes the identification number of the requesting IPU 1307. Subsequently, the result from the result register of the target resource is presented to all requesting processing resources capable of making requests 1309 by, for example, broadcasting the result to the requesting processing resources. As the response is provided to all the IPUs capable of making requests, if a shared resource (e.g., an addressable memory space) is freed or released from a lock operation, any IPUs that are sleeping in wait for such a locked resource may awaken. Also as the result includes the IPU number of the originally requesting IPU, the results may be obtained and used by the respective requesting IPU.

Fig. 14 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of routing a result from a target processing resource (e.g., L2 cache 112 or MPU 114) to delegating processing resources (e.g., IPUs 102, 104, 106, 108). As has already been discussed in Fig. 13, the result is presented to all IPUs capable of making requests 1403. Upon receiving the result from the target processing resource, each of the delegating processing resources determines if its IPU number is equal to an IPU number in the presented result 1405. If the IPU number in the presented result is equal to the IPU number of the instant IPU meaning that the presented result is in response to the request from the instant IPU, the instant IPU is told to wake 1407. In one non-limiting example embodiment, the instant IPU is woken by receiving a signal from comparison circuitry, which compares the IPU number in the result and the instant IPU number. The IPU numbers may be hardwired into each of the IPUs so as to differentiate multiple IPUs from one another. Upon being instructed to wake 1407, the result from the target processing resource is loaded into the result pipeline register of the instant IPU 1409. Thereafter, the instant IPU may execute the next instruction 1411.

If the IPU number in the presented result is not equal to the IPU number of the instant IPU 1405, a determination is made if an unlock bit/flag is set within the presented result 1413. If no such unlock bit/flag is set, the IPU does nothing and may keep on sleeping 1415. If an unlock bit/flag is set in the presented result 1413, then a determination is made regarding if the instant IPU is waiting for an unlock 1417. If the instant IPU is not waiting for an unlock instruction, the instant IPU does nothing and may keep on sleeping 1415. If it is determined that the instant IPU is waiting for an unlock 915, another determination is made whether the unlock address that is part of the presented result is equal to the locked address that is in the address register of the IPU 1419. If the unlock address that is part of the presented result is not equal to the locked address that is in the address register of the instant IPU, then the instant IPU will do nothing again and may keep on sleeping 1415. If it is determined that the two addresses (i.e., unlock and lock addresses) are equal 1419, then the IPU will be instructed to wake 1421. Upon waking, the IPU will retry executing the locked instruction that previously failed and caused its sleep 1423.

Such delegation of execution-instructions through Router 110 allows Processor 100 of the present invention to operate in an asynchronous or synchronous manner. As has been demonstrated above, various processing resources may wait for variable amounts of time and not interrupt the operation of other processing resources. Such flexibility in the execution of instructions results in significantly increased processing throughput and more efficient usage of power.

Sleep Lock

Fig. 15 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a sleep lock operation in L2 cache 112 of Processor 100. Initially, the L2 cache is presented with a sleep lock request from a requesting IPU via Router 110 1503. The request may be in a similar form constructed by the requesting IPU as shown in Fig. 11. As the cache may be presented with several other routed requests from requesting resources, the L2 cache (i.e., data optimizing cache) may check the target address that is provided in the presented request to determine if the address is locked based on its

locked variable 1505. For example, the lock-variable of negative numbers (e.g., -1) may indicate that the address is read/write locked by another request. A lock-variable of zero may indicate that the address is unlocked and available for any type of operation (e.g., read/write or read-only). A lock-variable of positive numbers (e.g., 1, 2, 3...) may indicate that the address is available only for a read-only operation. Subsequently, it is determined that if the lock-variable is less than zero 1507. If the lock-variable is less than zero (e.g., -1), that means the address is read/write locked by another request and the request may be discarded 1509. The requesting IPU may try again in a later cycle for the same address. If the lock-variable is not less than zero (e.g., 0, 1, 2, 3...), the L2 cache decodes the opcode of the received request 1511, and further determines whether the opcode indicates a read/write lock operation 1513.

If it is determined that the opcode does not indicate a read/write lock operation (i.e., a read-only operation), L2 cache 112 adds one to the lock-variable 1515 and subsequently performs a read-only operation by reading shared data at the address 1517. The L2 cache may return the performed result to the requesting IPU by adding the identification number of the requesting IPU. Upon completing the read-only operation, the L2 cache subtracts from the value of the lock-variable by one 1519. The L2 cache then checks the lock-variable again 1521 and it is determined whether the lock-variable is zero 1523. If the lock-variable is zero (i.e., no other pending read-only requests exist for the address), the L2 cache may release the address 1525 for all type of operation (e.g., read/write operation etc). The L2 cache may then continue for other operations such as going back to 1503 receive further requests.

If it is determined that the opcode indicates the read/write lock operation is required 1611, L2 cache 112 may check the lock-variable 1529 and it is determined if the lock-variable is zero 1531. If the lock-variable is not zero indicating the address is read-only locked by another request, the L2 cache may simply discard the request 1533. If the lock-variable is zero indicating that the address is available, the L2 cache may set the lock-variable as minus one 1535. The L2 cache then may perform the read/write lock operation by reading/writing the shared data at the address 1537. The L2 cache may return the

executed result to the requesting IPU by adding the identification number of the requesting IPU. Upon completing the read/write lock operation, the L2 cache may set the lock-variable back to zero 1539 and release the address 1525. The L2 cache may then continue for other operations such as going back to operations 1503 to receive further requests.

The following Table 1 further illustrates exemplary sequences to execute the sleep-lock operations in an address A of L2 cache 112 with requests from multiple IPUs (i.e., IPU 102, IPU 104, IPU 106) of Processor 100. It is assumed that the priority of the IPUs is in the order of IPU 102, IPU 104 and IPU 106 from highest to lowest, respectively.

Table 1.

Sequence	Lock-variable of address A	Request of IPU102	Request of IPU 104	Request of IPU 106
1	0	Read-only lock		
2	1	Read data	Read-only lock	Read/write lock
3	2	Unlock	Read data	
4	1		Unlock	
5	0			Read/write lock
6	-1	Read/write lock	Read-only lock	Read/write data
7	-1			Unlock
8	0	Read/write lock	Read-only lock	
9	-1	Read/write data		
10	-1	Unlock		
11	0		Read-only lock	
12	1		Read data	
13	1		Unlock	
14	0			

At sequence 1, a request from IPU102 is received at the L2 cache requesting a read-only lock to the address A of the L2 cache. The request is accepted because the lock-variable of the address A is zero at the cycle. IPU102 may be put to sleep after supplying the request.

At sequence 2, the L2 cache adds one to the lock-variable for the request from IPU 102 and reads the shared data from the address A for IPU 102. The lock-variable is now one indicating that there is one read-only lock request, which is pending in address A of the L2 cache. Simultaneously, two requests from IPU 104 and IPU 106 are received respectively requesting a read-only lock and a read/write lock to the address A. Among the two requests, only the request from IPU 104 (i.e., a read-only lock) is accepted either because IPU 104 has a higher priority than IPU 106 or a read/write lock request cannot be accepted with the positive value of the lock-variable (i.e., 1). IPU 106 may wait (i.e., sleep) until the address A is released and try again by issuing a new sleep-lock request upon waking up.

At sequence 3, the L2 cache adds one to the lock-variable for IPU 104 and reads the shared data for IPU 104. The lock-variable is now two indicating that two read-only lock requests are currently pending in address A of the L2 cache. The read-only lock for IPU 102 is completed at sequence 3.

At sequence 4, the L2 cache subtracts one from the lock-variable indicating that the request from IPU 102 is completed. The lock-variable is now one indicating that only one read-only lock request is currently pending at the address A of the L2 cache. The read-only lock for IPU 104 is completed at sequence 4. The L2 cache may broadcast signals using a dedicated return path to the sleeping IPUs (e.g., IPU 106) to inform that address A is unlocked.

At sequence 5, the L2 cache subtracts one from the lock-variable indicating that the request from IPU 104 is completed. The lock-variable is now zero indicating that there is no pending request for the address A. Simultaneously, a read/write lock request is received from IPU 106 and accepted because the lock-variable is zero.

At sequence 6, the L2 cache set the lock-variable to -1 indicating that the address is now locked by a read/write lock request. The L2 cache reads/writes the shared data at the address A. Simultaneously, a read/write lock request from IPU102 and a read-only request from IPU 104 are received, but these requests are not accepted because the address A is now locked by a read/write lock request. IPU 102 and IPU 104 may wait (i.e., sleep) until the address A is released and try again by issuing a new sleep-lock request upon waking.

At sequence 7, the read/write lock request from IPU 106 is completed. The L2 cache may broadcast signals using a dedicated return path to the sleeping IPUs (e.g., IPU 102 and IPU2 104) to inform that address A is unlocked.

At sequence 8, the L2 cache sets the lock-variable back to zero because the read/write lock request is completed at sequence 7. Simultaneously, a read/write lock request from IPU 102 and a read-only request from IPU 104 are received at sequence 8. Among the two requests, the read/write lock request from IPU 102 is accepted because IPU 102 has a higher priority than IPU 104. IPU 104 may wait (i.e., sleep) until the address A is released and try again by issuing a new sleep-lock request upon waking.

At sequence 9, the L2 cache sets the lock-variable to -1 for the read/write request from IPU 102 and reads/writes data from address A of the L2 cache.

At sequence 10, the read/write lock request from IPU 102 is completed. The L2 cache may broadcast signals using a dedicated return path to the sleeping IPUs (e.g., IPU 104) to inform that address A is unlocked.

At sequence 11, the L2 cache sets the lock-variable back to zero because the read/write lock request is completed at sequence 10. Simultaneously, a read-only lock request from IPU 104 is received and accepted.

At sequence 12, the L2 cache adds one to the lock-variable and reads shared data from address A for IPU 104.

At sequence 13, the read-only lock request from IPU 104 is completed. The L2 cache may broadcast signals using a dedicated return path to sleeping IPUs, if any, to inform that address A is unlocked.

At sequence 14, the L2 cache subtracts one from the lock-variable indicating that the read-only request from IPU 104 is completed. The lock-variable is now zero indicating that address A is unlocked.

As described above, a sleep-lock operation at L2 cache 112 allows multiple requesting processing resources (e.g., IPUs) to share data effectively in 'critical regions' of application programs. That is, requesting IPUs may sleep if another requesting IPU has the sleep-lock. The sleeping IPUs may wake upon receiving signals from the L2 cache informing that the specific address that the requesting IPUs want to access is unlocked. The sleep-lock operation may be good for relatively short (i.e., a low level) and/or critical regions of application programs. For long (i.e., a high level) critical regions, instead of using the sleep-lock as described above, an alternative process may be used. For example, if another requesting IPU has a semaphore, rescheduling of processing tasks is performed by the operating system and another rescheduling is performed upon release of the semaphore. The sequences of the lock-variable for the rescheduling is similar to the sequences of the sleep-lock.

Cache Coherence

As the requesting IPUs perform either the sleep-lock operation or the semaphore operation, the values between the internal caches and the external cache may be different because the internal caches tend to be updated by the requesting IPUs, i.e., the caches are incoherent.

Fig. 16 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of maintaining the cache coherence between the internal caches (e.g., L1 caches 103, 105, 107, 109) and the external cache (e.g., L2 cache 112).

An IPU that performs either a sleep-lock operation or a semaphore operation checks the responses from the external cache 1603 and it is determined whether the operation (i.e., a sleep-lock or semaphore operation) is completed 1605. If it is determined that the operation is not completed, the IPU keeps checking until the operation is completed 1603. If it is determined that the operation is completed, the IPU checks the global area of its internal

cache 1607. A determination is then made whether data in the global area of the internal cache is updated by the operation 1609. If it is determined that the data is updated, the IPU requests the external cache to copy the updated data into the external cache's global area 1611. As a result of the copy at the end of the operation, it is guaranteed that a next requesting IPU will access updated data, not old data. If it is determined that the data is not updated 1709, the data are simply discarded by the IPU 1613.

Optimizing Cache

Fig. 17 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of a data optimizing cache facility in L1 caches 103, 105, 107, 109 and L2 cache 112 of Processor 100.

While the basic concept of the data optimizing in the caches is similar throughout the caches, the exemplary method assumes that the optimizing process is performed in the L2 cache, which is external to the IPUs. The L2 cache may be presented with a request from a requesting IPU 1703. The request may be in a similar form constructed by the requesting IPU as shown in Fig. 11. The L2 cache reads the request to get an address that the requesting IPU wishes to access and happens to be available in the L2 cache 1705. The data optimizing cache may include logic circuitry to make comparisons based on targeted address. In one non-limiting example embodiment, the data optimizing cache compares the target address with address values representing a stack address base and end point, which may be maintained in base and end registers, and allocated by, for example, the operating system (e.g., Linux). The address range may be maintained in the base and end registers of the L2 cache. The data optimizing cache then determines whether the data requested is global or local 1707.

If the target address value falls within the values held in the base and end registers, values from the target address will then be deemed local. When a target address is deemed local, the data optimizing cache will get data at the target address 1709. Subsequently, as the data from the target address is obtained, the data optimizing cache may use a hash function to place retrieved data into a "local" segment of the cache 1711.

If the data optimizing cache finds that the target address is not within address values of the base and end of a stack 1707, the target address is deemed to be global. The data optimizing cache then get data at the target address from the L2 cache or memory 1713. The L2 cache then may use, for example, a hash compliment to place the retrieved data from the target address into a global region of the optimizing cache 1715. Upon placing a retrieved value into an appropriate region of the cache 1711, 1715, the cache may continue processing 1717.

As such, in this example embodiment, the hash function determines where in the data optimizing cache a retrieved value is to be stored and the retrieved value will be placed either in a local or a global region of the data optimizing cache based on the address. As such, the comparison between the targeted address and predetermined address range determines whether the target address is local or global, and the hash function further determines where in a cache a value is to be stored (i.e., a local or global region). It should be noted that based on the hash function, data may be categorized and saved in specified and desired portions and allotments of the cache memory. In one example, a hash function may be used to divide the available cache memory equally between local and global values. For example, if local, the highest hash bit will be 1, and otherwise 0 for global.

In an alternative embodiment, a hash function may be used to provide more memory space within the data optimizing cache for local values than global values. By providing more memory within the data optimizing cache to local values, a greater hit rate may be achieved by the cache as programs tend to cycle with locality of reference. It should be noted that the hash function may be altered and used to provide optimization based on various data processing needs. For example, a hash function may be used to partition the memory space of a data optimizing cache into three segments global instruction, global data and local data. In yet another alternative example embodiment, a hash function may be provided to partition the memory space of data optimizing cache into four regions including global instructions, global data, local instructions, and local data. It should be noted that the

hash function of the data optimizing cache may be altered to provide numerous and limitless data and category regions based on particular processing needs.

INSTRUCTION SET ARCHITECTURE

Figs. 18(a)~(e) are of functional block diagrams illustrating non-limiting example, inventive aspects of the present disclosure providing a format of instructions designed for Processor 100. A 16-bit fixed size instruction set is designed for Processor 100 of the present invention as an embodiment. Appendix 1 illustrates an exemplary 16 bit instruction set for a 32 bit system architecture. Appendix 2 illustrates an exemplary 16 bit instructions set for a 64 bit system architecture.

Fig. 18(a) is a literal prefixed instruction with a 12-bit literal prefix, Fig. 18(b) is an optionally prefixed instruction with a 5-bit literal prefix, Fig. 18(c) is an optionally prefixed instruction with an 8-bit literal prefix, Fig. 18(d) is an always prefixed instruction with an 8-bit literal prefix, and Fig. 18(e) is a never prefixed instruction with no literal prefix, respectively. Using the instruction set, small literal constants may be embedded in an instruction, and the size of the literal constant may be extended with the literal prefix instruction, each of which extends a literal by 12 bits in this embodiment. Alternatively, a different number of bits for a literal constant and literal prefix may be employed depending on deployment requirements. This instruction set structure allows for optimization in processor operation and caching as will be seen throughout the disclosure.

Literal prefix and literal constant functionality allows for instruction operand reduction and instruction extension. Such instruction size reduction allows for eased and more efficient cache coherency maintenance by resulting in 16-bit opcodes. Generally, many instructions offered by the present invention may be preceded by the literal prefix instruction (LPX) (e.g., Fig. 18(a)) with 12 extra literal bits, which serve to extend the literal data or address ranges of the instructions. For example, it is possible to generate 32 bit value of data or address with an instruction with 8 bit literals (e.g., Fig. 18(c)) by concatenating the two 12 literal bits from two LPX instructions before the instruction with 8 bit literals on which they are to act. As further examples, the optionally prefixed instruction with 5-bit literal with two

LPX instructions will give 29 bits of literal (i.e., 5bit + 2x12 bits) and instructions with an address literal (i.e., Fig. 18(d)) are always prefixed by one or two LPX instructions, which extend the address literal to 20 (i.e., 8+12) or 32 (8+2x12) bits.

The literal prefix is an instruction in its own right, and executed in the IPU pipeline like any other instructions. In the operation, the IPU and the literal prefix instructions may be configured in such a way that one or more literal prefix instructions (i.e., Fig. 18(a)), upon being fetched into the instruction registers of the IPUs, sets the flag of a status register implying that an optionally and/or always prefix instruction (i.e., Figs. 18(b)~(d)) or another literal prefix instruction will be followed. The optionally and/or always prefixed instruction, upon being fetched into the appropriate registers by the IPUs, clears the prefixed flag set by the literal prefix instructions. The fetched one or more literal prefix instructions and the optionally and/or always prefix instruction may be executed simultaneously. In an embodiment, instructions are fetched by the IPUs in 64 bit words of four instructions expediting the fetching process. As a result, the pipeline of the IPUs may be designed in such a way that an instruction and the literal prefix instructions are simultaneously executed as long as they are within the same 64 bit word. With additional hardware, they may also be simultaneously executed even if they are in adjacent 64 bit words, i.e., not within the same 64 bit word.

Instructions with fixed size 16 bits have been limited to a subset of the instruction set that is required to effectively process all C/C++/Java data types and operations, allowing one instruction to do that which other less focussed architectures require a sequence of instructions to perform. This means that the 16 bit instruction size architecture leads to short compiled machine code for a given source program. This not only saves on the amount of memory required to store programs, but also leads to faster execution because a 64 bit memory word fetch brings in more instructions at a time. Moreover, by adopting the 16 bit fixed size instruction, the instruction decoding logic is relatively simple and fast, and the instruction execution pipeline is also short, simple and fast. This simultaneously leads to high silicon efficiency and high speed execution without the need for using expensive

techniques such as long pipelines which have a high stall cycle penalty on every program branch, instruction reordering and speculative execution which uses a lot of silicon area and slows execution down by almost as much as it gains, and/or a branch prediction which also uses a lot of silicon area and is needed for long pipelines but not short pipelines. However, despite obviously being a good idea, to fit every required data type and operation in a 16 bit fixed size instruction set has been difficult for designers.

To fit every required data type and operation within a 16 bit fixed size instruction, the following three methods have been developed and used in the present invention.

(1) Literal prefix as a 17^{th} bit of the 16 bit instructions

There exist three types of instructions in the present invention, i.e., instructions that are optionally literal prefixed (e.g., Figs. 18(b), 18(c)), instructions that are always literal prefixed (e.g., Fig. 18(d)), and instructions that are never literal prefixed (e.g., Fig. 18(e)). Where two instructions co-exist, one of which always requires a literal prefix and the other forbids it, they may share the same instruction opcode without ambiguity. This extends the opcode space of the instructions and thereby increases the number of instructions that can be encoded in the 16 bit instruction format. The presence of a literal prefix is indicated by a flag bit in the IPUs' status register, which effectively acts as a 17^{th} opcode bit of the 16 bit instructions making each processor essentially a 17 bit instruction size.

(2) Reduced bits to represent registers

Each of IPUs 102, 104, 106, 108 of Processor 100 includes sixteen registers each with thirty-two bits in an embodiment. The registers have been classified into two different types, i.e., 8x32 bit data registers and 8x32 bit address registers. It is configured so that only three bits of a 16 bit instruction are required to represent a register of the 16 registers. The rest of one required bits is implied by the opcode of the instruction, i.e., type of register is determined by decoding the opcode of the instruction.

(3) Instructions with different number of opcode

Only the highest frequency instructions have three operand, most other instructions have two operand and some with one operand.

Fig. 19 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of literal prefix and literal constant functionality. At some point, an IPU fetches an instruction 1901. The IPU determines if a literal flag is set by examining a special register 1903. The special register may be in the IPU status register. If the literal flag has been set, then the IPU determines if there is a literal prefix (e.g., 00) 1905. If a literal prefix is present, then the IPU will append in literal constant in the literal register (e.g., by employing a shift) 1907. Upon appending the literal constant, the IPU may continue instruction fetch iteration 1901. If there is no literal prefix in the fetched instruction 1905, the IPU will execute the opcode as an extended operation with a constant in the literal register 1909. Upon extended execution of the opcode, the IPU may continue instruction fetch iteration 1901.

If no literal flag is set 1903, the IPU determines if a literal prefix is present in the fetched instruction 1911. If no literal prefix is supplied, the IPU may execute the opcode in the fetched instruction in a non-extended manner 1917. However, if a literal prefix is provided in the fetched instruction 1911, then the IPU may set a literal constant (LC) flag in a status register 1913. Upon setting the literal constant flag, the IPU may put the literal constant from the literal prefix instruction into the literal register 1915. Upon loading the literal register 1915 or executing the opcode 1917, the IPU may continue instruction fetch iteration 1901.

Context Switch Accelerator

The reduction of operands into 16-bit instructions ensures that such instructions are always bit aligned. As such, it is impossible that an instruction register will ever point to an odd address. Thus, only the last 31 bits in the instruction register are required to address a desired address, which provides an extra bit in the instruction register to be used for other purposes. Thus, a context switch accelerator may be enabled by using the otherwise nugatory 32nd bit in the instruction register. A nugatory bit may be used to determine if

Processor 100 is in system mode (e.g., 0) or in user (e.g., 1) mode. By using the 32nd bit for system or user status, Processor 100 is obviated of the need to save status registers or return pointer values as is required in conventional processors. Thus, as the context switch accelerator need only use the instruction register, the requirements to fulfill a context switch are reduced to and become similar to a normal subroutine call. Such context switch acceleration is made possible because Processor 100 no longer needs to flush and store additional registers, yet Processor 100 still properly allows user/system mode access while requiring only one cycle (i.e., the same instruction pointer and jump requirements of a normal subroutine call). Normally, a cycle is required for every register save in a context switch, therefore the context switch accelerator saves cycles by only having to store the instruction register. The prior art employs multiple cycles and uses more memory.

BINDING ACCELERATION

Fig. 20 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a late/dynamic binding accelerator. In one exemplary embodiment, all late/dynamic binding names are stored at an odd address by a linker 2001. This may be achieved by writing a compiler to store such late/dynamic binding names and employ an odd address by simply adding 1 in the 32^{nd} bit to any address for the binding names. As such, if an IPU attempts to load an instruction register with an odd address value 2003, the IPU generates a binding interrupt signal 2005. If the IPU attempts to load the instruction register with an even value 2003, no interrupt is required and processing may continue 2019. Upon generating the interrupt 2005, the operating system may interpret the late/dynamic binding interrupt (i.e., vectoring to interrupt) 2007. Upon interpretation, the operating system performs a look-up on the odd address returned with the interrupt 2009. The late/dynamic binding accelerator then replaces the odd address in the instruction register stored in the stack with an even address found in the look-up of the late/dynamic binding names table that was established by the linker 2011. The even address found in the look-up is the actual address for the code that is to be executed. The late/dynamic binding accelerator then determines if late binding is being used by examining the linkage table entry 2013. If

late binding is being employed, then the odd address is replaced with even instruction pointer stored in stack 2015. If dynamic binding is being employed, then the odd address is replaced with an even instruction pointer in the linkage table entry 2017, so that there will be no interrupt on a later access.

MULTICYCLE INSTRUCTION REDUCTION

Status Register Cycle Flags

Fig. 21 is of a table illustrating one, non-limiting example, inventive aspect of the present disclosure of multicycle instruction reduction. The table illustrates that the value of a cycle flag is set depends upon the number of cycles that will be required based on an operation. Thus, for the operations that require a simple (e.g., normal) cycle to complete (2109), the operations will result in the cycle flags being set to "0" (2102). Operations requiring two cycles or double processes (2111) such as a load or save operation will set the cycle flags to "1" (2103). Multi-cycle/double processes (e.g., extended) operations (2113) will set the cycle flag to "2" (2105). Repeated operations (2115) such as multi-cycle, single or double process status operations will set the cycle flag to "3" (2107).

Register Address Reduction

Status Register Type Flag

Fig. 22 is of a table illustrating one, non-limiting example, inventive aspect of the present disclosure which provides various IPU registers, which may be accessed based on type flags that are set in a reduced register address portion of an instruction. In one example embodiment of Processor 100 employing 16 registers, only 3 dedicated bits are required to address all 16 registers. Normally, addressing 16 registers would require 4 bits of address. Such register address instruction reduction is made possible by dedicating 3 bits to register addressing. Additional 1 bit come from the interpretation of an operation code, which sets type flags in a status register as shown in the table. Thus, based on the type flag, various register types (e.g., data type (2213) or address type (2215)) may be accessed. In this example embodiment, the registers may hold 32 bits of data 2209, 2211. In alternative

embodiments, the registers may be of variable size (e.g., 48 bits, 64 bits, 96 bits, 128 bits, etc.). As 3 bits are dedicated to register addressing, each type of register bank may itself contain up to 8 registers 2205, 2207. Thus, when the type flag in a status register is set to "0" 2201, the 3 bits dedicated to register addresses will access data registers. When the type flag is set to "1" 2203 in combination with the dedicated 3 bits to register addresses, the address registers may be accessed. As such, the type flags being set in a status register in combination with 3 bits dedicated to register addressing allows for access to a total of 16 registers. Such register address bit reduction allows reduced instructions (e.g., an instruction set that may be fit into 16 bits, which increases cache coherency and overall processor efficiency).

ATOMIC INSTRUCTIONS

Fig. 23 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing an event sleep facility. As some point, an even sleep instruction is fetched 2301 and the instruction is used to set an IPU to sleep until an event occurs 2303. An event sleep instruction is an atomic instruction in a processing cache (e.g., L2 cache 112). The processing cache determines if an event variable stored in the processing cache is set to zero 2305. If the event variable is not zero 2903, then the processing cache sends a reply to an IPU bus with the event variable address, which will wake the appropriate IPU from sleep 2307. If the event variable is equal to zero 2305, then the processing cache will set the IPU to sleep until event variable is set to a nonzero value 2311. Thereafter, the processing cache may retry the event sleep instructions 2301. Upon sending the reply 2307, the processing cache may continue to operate 2309.

Fig. 24 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing an event set instruction facility. An event set instruction is an atomic operator in a processing cache (e.g., L2 cache 112). At some point, an event set instruction is fetched 2401 and used to notify a processing resource that an event is taking place 2403. A parameter is combined with an event variable through a logical-OR operation in the L2 cache 2405. The parameter is a bit mask and is obtained from the data being

written to the L2 cache. If the resulting event variable is equal to zero 2407, then the processing cache will continue to operate 2409. If the resulting event variable is not equal to zero 2407, then the data optimizing cache sends a reply to the IPU bus with the event address and value, which wakes the sleeping IPU that is waiting on the event 2411. Upon sending the reply, the data optimizing cache may continue to operate 2413.

In one exemplary embodiment, each IPU can track up to 32 events, which is limited by the word length of an IPU, and which may operate similarly to interrupts as has already been discussed.

Wait-On-Semaphore

Fig. 25 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a wait-on-semaphore instruction facility. A wait-on-semaphore instruction is an atomic operation for a processing cache (e.g., L2 cache 112). At some point, a wait-on-semaphore instruction is fetched 2501 and used to claim exclusive use of some shared resource or memory 2503. Upon interpreting a wait-on-semaphore instruction operation 2505, the data optimizing cache will subtract "1" from a semaphore variable in the L2 cache 2507. If the processing cache determines that the semaphore variable is negative 2509, an operating system trap-call to wait on the queue is issued 2513. This trap-call causes rescheduling so that other program threads/processes may wait for execution of a thread to release its semaphore on a particular IPU. If the semaphore variable is not negative 2509, the processing cache may continue to operate 2511.

Signal-On-Semaphore

Fig. 26 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a signal-on-semaphore instruction facility. A signal-on-semaphore instruction is an atomic operation of a processing cache (e.g., L2 cache 112). At some point, a signal-on-semaphore instruction is fetched 2601 and used to release some shared resource or memory from exclusive use 2603 (e.g., the signal-on-semaphore instruction is used to signal that a processing resource is no longer being used). In order to

synchronize the L1 caches of IPUs 102, 104, 106, 108 and L2 cache 112 prior to the semaphore operation, the global data in the L1 caches may be checked with a hash function 2605. Subsequently, it is determined whether the global data in the L1 caches are dirty (i.e., updated during an exclusive operation) 2607. If the global data are dirty, they are flushed to the L2 cache by writing all dirty L1 cache global data to the L2 cache 2609 and marking all L1 cache global data entries empty 2611. If the global data are not dirty 2607, then operation at 2609 is skipped. The processing cache then adds "1" to a semaphore variable in the L2 cache 2613. Subsequently, it is determined whether the semaphore variable is positive 2615. If a semaphore value is positive, then the data optimizing cache will continue to operate 2617. If the semaphore variable in the L2 cache is not positive 2615, an operating system trap-call is issued to signal on the semaphore wait queue 2619). Such an operating system trap signal has the effect of rescheduling so that the IPU may execute other programs/threads/processes.

Sleep Lock Instruction

Fig. 27 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a sleep lock instruction facility. A sleep lock instruction is an atomic L2 cache operation. At some point, a requesting IPU (e.g., IPU 102) issues an access request to shared memory variables in the processing cache (i.e., L2 cache 112) 2701. The L2 cache determines the type of lock desired based on the access request 2703. If the desired lock type is a read/write operation, the L2 cache then determines if the lock variable is "0" (i.e., unlocked) 2705. If the lock variable is zero, then the L2 cache sets the lock variable to "–1", and further sets the lock flag of the status register in the requesting IPU 2707. Upon setting the lock variable, the L2 cache continues to process 2709. If the lock variable is not "0" 2705 (e.g., "-1"), then the L2 cache will instruct the requesting IPU to sleep until an unlock is performed with the same lock variable and then retry the lock instruction 2711. If the desired type of lock is a read-only lock 2703, then the L2 cache will determine if the lock variable is negative 2713. If the lock variable is negative, then the L2 cache will instruct the IPU to sleep until it is unlocked with the same lock variable and then

retry the lock instruction 2711. If the lock variable is not negative 2713, then the L2 cache adds "1" to the lock variable of the L2 cache and sets the lock flag of the status register in the requesting IPU 2715. Upon adding to the lock variable, the L2 cache may continue to operate 2709.

Sleep Unlock Instruction

Fig. 28 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing an unlock operation. As has already been discussed, an IPU may receive a response from the data processing cache (i.e., L2 cache 112) 2801. The IPU upon receiving the response determines whether it wishes to free access to an area of the L2 cache 2803. The IPU then issues an unlock instruction directed at the lock variable of the target address in the received response 2805, which will be further discussed in Fig. 29.

Fig. 29 continues to illustrate the unlock operation from Fig. 28 2902. The IPU determines value types (e.g., global or local) by using, for example, a hash function 2904. The IPU then determines if each global value in the L1 data cache is hot or not 2906. A hot global value is one that has been loaded to the L1 cache (e.g., L1 cache 103) while the lock was set. If the global value in the L1 cache is not hot 2906, a coherence check is performed 2908 as is further discussed in Fig. 30. If a global value in the L1 cache is hot 2906, the IPU then determines if that value is dirty 2912. A dirty value is one where the value in the L1 cache has not been updated to the L2 cache. If the global value in L1 is not dirty 2912, the value in the L1 cache is dropped 2914 without requiring the IPU to do any further work and execution may continue 2910. If the global value in the L1 cache is dirty 2912, the global value is written out from the L1 cache to the L2 cache 2916. After writing from the L1 to the L2 cache, the cached address is then marked as being free 2918. Thus, upon ensuring that the cache is coherent 2914, 2918, processing may continue 2910.

Fig. 30 continues to illustrate an unlock operation in ensuring the cache coherence 2908 of Fig. 29 3002. To ensure the cache coherence, "1" is subtracted from the locked variable of the target address in the L2 cache 3004. Subsequently, a determination is made if the target address is still locked by checking the lock variable of the target address (e.g., if

the target address lock variable is greater than zero, it is still locked) 3006. If the lock value is still greater than zero, the target address and the lock value of the target address are sent as a reply to the IPU bus 3010. In the case where the lock value is greater than zero 3006, multiple processing resources, e.g., IPUs, may be reading/accessing the target address, and as such the remaining reading resources are informed and allowed to continue accessing the target address while a read-only lock is maintained. If the lock variable of the target address is less than or equal to zero 3006, the lock variable is set to zero 3008 to tidy up. Upon setting the lock variable to zero (i.e., unlocking the target address), the L2 cache sends the target address and the lock value along the IPU bus to inform the IPUs 3010. Upon providing the target address and lock value, processing may continue 3012.

Other atomic instructions in the L2 cache include the ability to add a parameter to a value in the L2 cache, to perform a logical-OR of a parameter to a value in the L2 cache, to perform a logical-AND to a parameter to a value in the L2 cache, and to perform a logical-NOT to a parameter to a value in the L2 cache. All of the above-noted atomic instructions of the L2 cache may be accessed from C macros and/or other higher language functions. It should be noted that the decoding of an opcode may cause delegation of operations to the L2 cache from an IPU or some other processing resource. It should be further noted that the L2 cache instruction set is but one example set, and that numerous and varied alternative embodiments may be employed.

It should be understood that the above description is only representative of illustrative embodiments. For the convenience of the reader, the above descriptions have focused on a representative sample of various possible embodiments, a sample that teaches the principles of the invention. The description has not attempted to exhaustively enumerate all possible variations. That alternate embodiments may not have been presented for a specific portion of the invention or that further undescribed alternate embodiments may be available for a portion is not to be considered a disclaimer or abandonment of those alternate embodiments. It will be appreciated that many of those undescribed embodiments incorporate the same principles of the invention and others are equivalent. Thus, it is to be understood that the

embodiments and variations shown and described herein are merely illustrative of the principles of this invention and that various modifications may be implemented without departing from the scope and spirit of the invention.

In addition, the disclosure herein includes other inventions not presently claimed. Applicant reserves all rights in those presently unclaimed inventions including the right to claim such inventions, file additional applications, continuations, continuations in part, divisions, and/or the like thereof.

CLAIMS:

1. A synchronicity independent, inter-resource and intra-processor execution-instruction delegating processor apparatus, comprising:

a processing resource;

an execution-instruction signal router, wherein the instruction signal router is disposed in communication with the processing resource;

an other processing resource, wherein the other processing resource is disposed in communication with the execution-instruction signal router;

wherein the processing resource delegates processing execution-instruction signals to an other processing resource through the execution-instruction signal router.

2. The apparatus of claim 1, wherein a processing resource is an integer processing unit.

3. The apparatus of claim 1, wherein a processing resource is a mathematical processing unit.

4. The apparatus of claim 1, wherein a processing resource is a memory management unit.

5. The apparatus of claim 1, wherein a processing resource is a vector processing unit.

6. The apparatus of claim 1, wherein a processing resource is a digital signal processing unit.

7. The apparatus of claim 1, wherein a processing resource is a graphics processing unit.

8. The apparatus of claim 1, wherein a processing resource is an input/output controller processing unit.

9. The apparatus of claim 1, wherein a processing resource is an execution-instruction processing cache.

10. The apparatus of claim 1, wherein the execution-instruction signal router is a cross-point switch.

11. The apparatus of claim 1, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

12. The apparatus of claim 1, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

13. The apparatus of claim 1, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

14. The apparatus of claim 1, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

15. The apparatus of claim 1, wherein processing resources are communicatively disposed on a same die.

16. The apparatus of claim 15, wherein the execution-instruction signal router is on the same die with processing resources.

17. A synchronicity independent, inter-resource and intra-processor execution-instruction delegating processor apparatus, comprising:

a plurality of processing resources;

a memory, wherein the memory is communicatively accessible by the processing resources, wherein the memory may be accessed simultaneously by the processing resources, wherein the memory includes an instruction unit;

wherein the plurality of processing resources and memory are on the same die.

18. The apparatus of claim 17, wherein a processing resource is an integer processing unit.

19. The apparatus of claim 17, wherein a processing resource is a mathematical processing unit.

20. The apparatus of claim 17, wherein a processing resource is a memory management unit.

21. The apparatus of claim 17, wherein a processing resource is a vector processing unit.

22. The apparatus of claim 17, wherein a processing resource is a digital signal processing unit.

23. The apparatus of claim 17, wherein a processing resource is a graphics processing unit.

24. The apparatus of claim 17, wherein a processing resource is an input/output controller processing unit.

25. The apparatus of claim 17, wherein a processing resource is an execution-instruction processing cache.

26. The apparatus of claim 17, wherein the execution-instruction signal router is a cross-point switch.

27. The apparatus of claim 17, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

28. The apparatus of claim 17, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

29. The apparatus of claim 17, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

30. The apparatus of claim 17, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

31. The apparatus of claim 17, wherein processing resources are communicatively disposed on a same die.

32. The apparatus of claim 31, wherein the execution-instruction signal router is on the same die with processing resources.

33. The apparatus of claim 17, wherein the instruction unit an instruction determination unit.

34. The apparatus of claim 17, wherein the instruction unit an instruction execution unit.

35. The apparatus of claim 17, wherein the plurality of processing resources and the memory are communicatively connected through an execution-instruction signal router.

36. A processing cache memory apparatus, comprising:

a memory;

an instruction determination unit,

wherein the memory is disposed in communication with the instruction determination unit;

wherein the instruction determination unit examines supplied signals to identify instructions and stores values into a location in the memory within a single cycle;

an instruction execution unit,

wherein the memory is disposed in communication with the instruction execution unit;

wherein the instruction execution unit executes any identified instructions and stores values into a location in the memory within a single cycle.

37. The apparatus of claim 36, wherein the instruction determination unit and the instruction execution unit are a singular unit.

38. The apparatus of claim 36, wherein the memory, instruction determination unit, instruction execution unit are on the same die.

39. The apparatus of claim 36, wherein the instruction determination unit and the instruction execution units are capable of determining and executing an event-sleep command.

40. The apparatus of claim 36, wherein the instruction determination unit and the instruction execution units are capable of determining and executing an event-set command.

41. The apparatus of claim 36, wherein the instruction determination unit and the instruction execution units are capable of determining and executing a wait-on-semaphore command.

42. The apparatus of claim 36, wherein the instruction determination unit and the instruction execution units are capable of determining and executing a signal-on-semaphore command.

43. The apparatus of claim 36, wherein the instruction determination unit and the instruction execution units are capable of determining and executing a sleep-lock command.

44. The apparatus of claim 36, wherein the instruction determination unit and the instruction execution units are capable of determining and executing a sleep-lock-destruction command.

45. The apparatus of claim 36, wherein the instruction determination unit and the instruction execution units are capable of determining and executing a add command.

46. The apparatus of claim 36, wherein the instruction determination unit and the instruction execution units are capable of determining and executing a logical-OR command.

47. The apparatus of claim 36, wherein the instruction determination unit and the instruction execution units are capable of determining and executing a logical-AND command.

48. The apparatus of claim 36, wherein the instruction determination unit and the instruction execution units are capable of determining and executing a logical-NOT command.

49. The apparatus of claim 36, wherein a processing resource may sleep while an other processing resource executes execution-instructions.

50. The method of claim 49, wherein a sleeping processing resource may be woken by a response to a request.

51. The method of claim 50, wherein the response is directed to a specific processing resource.

52. The method of claim 50, wherein the response indicates a once locked resource is unlocked.

53. The apparatus of claim 49, wherein the other processing resource is the memory.

54. The apparatus of claim 36, wherein a plurality of processing resources and the memory are communicatively connected through an execution-instruction signal router.

55. An inter-resource, intraprocessor execution-instruction delegation apparatus, comprising:

a resource arbitration unit,

wherein the resource arbitration unit selects a source processing resource from which to obtain an execution-instruction signal,

wherein the resource arbitration unit includes a priority bit comparator,

wherein the priority bit comparator is communicatively disposed with the resource arbitration unit

wherein the resource arbitration unit is communicatively disposed with processing resources.

56. The apparatus of claim 55, wherein the arbitration unit is a counter employing $\log_2$ bits a number of total processing resources to address processing resources.

57. The apparatus of claim 55, wherein the number of total processing resources is a number of processing resources interconnected through the resource arbitration unit within a single die.

58. The apparatus of claim 55, wherein the number of total processing resources is a number of processing resources within a single processor.

59. The apparatus of claim 55, wherein the resource providing an execution-instruction signal with the highest priority is selected by the resource arbitration unit.

60. The apparatus of claim 55, wherein the resource arbitration unit and the priority bit comparator obtain instructions and are communicatively disposed.

61. The apparatus of claim 55, further comprising,
an iterating dead-lock bit counter.

62. The apparatus of claim 61, wherein the resource arbitration unit selects a source processing resource's execution-instruction signal by employing the dead-lock bit counter.

63. The apparatus of claim 55, further comprising,

a resource delegation unit.

64. The apparatus of claim 63, wherein the resource delegation unit is communinicatively disposed with the resource arbitration unit.

65. The apparatus of claim 64, wherein the resource delegation unit directs a selected execution-instruction signal to a processing resource.

66. The apparatus of claim 64, wherein the resource delegation unit employs the selected execution-instruction signal's operation-code to identify a target processing resource.

67. The apparatus of claim 55, wherein the execution-instruction signal was selected by the resource arbitration unit.

68. The apparatus of claim 55, wherein the resource arbitration unit is a cross-point switch; wherein the resource arbitration unit has less than a 100 picosecond delay.

69. The apparatus of claim 55, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions through the resource arbitration unit.

70. The apparatus of claim 55, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

71. The apparatus of claim 55, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

72. The apparatus of claim 55, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

73. The apparatus of claim 55, wherein the resource arbitration unit is on the same die with processing resources.

74. The apparatus of claim 55, wherein a plurality of processing resources and a memory are communicatively connected through a resource arbitration unit.

75. A medium readable by a processor, comprising:

execution-instruction signals in the processor readable medium, wherein the execution-instruction signals are issuable by the processor and include:

a processing resource identifier, wherein the processing resource identifier identifies the origin of an execution-instruction signal;

an operation-code, wherein the operation code identifies a target processing resource;

a data field, wherein the data field may be used by a processing resource as an operand;

a priority field, wherein the priority field may be used to prioritize scheduling of an execution-instruction signal.

76. The medium of claim 75, wherein the processing resource identifier is log2 bits the number of available processing resources.

77. The medium of claim 75, wherein the operation-code is 10 bits.

78. The medium of claim 75, wherein the data field may contain an address.

79. The medium of claim 75, wherein the data field may contain information to be processed.

80. The medium of claim 75, wherein the data field is 64 bits.

81. The medium of claim 75, wherein the priority field is 2 bits.

82. The medium of claim 75, further comprising,
an address field.

83. The medium of claim 82, wherein the address field is 32 bits.

84. The medium of claim 82, wherein the address field employs literal prefixes.

85. A processor comprising:
an integer processing unit adapted to execute instructions of a program;
a math processing unit adapted to receive a request from the integer processing unit; and
a router that interfaces between the integer processing unit and the math processing unit, wherein the router is adapted to route the request from the integer processing unit to the math processing unit.

86. The processor according to claim 85, wherein the integer processing unit is adapted to execute instructions only with either an add calculation or a subtract calculation of integer numbers.

87. The processor according to claim 85, wherein the integer processing unit is adapted to detect an instruction with either a floating-point calculation, a multiple calculation or a divide calculation.

88. The processor according to claim 87, wherein the integer processing unit is adapted to extract an opcode from the instruction upon detection and forward the opcode to the math processing unit with a calculation request through the router.

89. The processor according to claim 87, wherein the integer processing unit is adapted to forward the instruction to the math processing unit with a calculation request through the router.

90. The processor according to claim 85, wherein the integer processing unit includes an internal cache and the internal cache is divided into a local and global areas thereby increasing a cache hit rate.

91. The processor according to claim 85, wherein the math processing unit is adapted to receive the request using a pipeline structure.

92. The processor according to claim 85, wherein the math processing unit is adapted to return a result in response to the request from the integer unit.

93. The processor according to claim 85, further comprising a second integer processing unit adapted to execute instructions either from one program or from another program.

94. The processor according to claim 85, further comprising a second math processing unit adapted to receive a request from the integer processing unit.

95. The processor according to claim 85, further comprising a cache unit which interfaces with the integer processing unit through the router, wherein the cache unit is adapted to receive a request from the integer processing unit.

96. The processor according to claim 95, wherein the cache unit is adapted to receive the request using a pipeline structure.

97. The processor according to claim 95, wherein the cache unit is divided into a local and global areas thereby increasing a cache hit rate.

98. The processor according to claim 95, wherein the cache unit is adapted to perform small set of atomic functions including a bus-lock control and a read-modify-write control.

99. The processor according to claim 95, wherein the cache unit is adapted to be connected directly to any of a memory, a test access port and a time divisional multiplexing data port.

100. The processor according to claim 99, wherein the test access port is a port for the Joint Test Action Group (JTAG) with a standard IEEE 1149.

101. A processor comprising:

a plurality of processing units each to execute instructions of a program independently; and

a cache unit configured to receive access requests from the plurality of processing units and returns results to the plurality of processing units in response to the access requests,

wherein each of the plurality of processing units is configured to sleep after sending an access request to the cache unit.

102. The processor of claim 101, wherein the cache unit is further configured to broadcast a signal to the plurality of processing units so that the plurality of processing units may receive the signal simultaneously.

103. The processor of claim 102, wherein the signal includes a requested result from one of the plurality of processing units combined with an identification information of the one of the plurality of processing units.

104. The processor of claim 103, wherein each of the plurality of the processing units is configured to detect the identification information thereby determining whether the identification information matches with its own identification information.

105. The processor of claim 104, wherein each of the plurality of the processing units is configured to wake up and continues operation with the requested result upon determining that the identification information matches with its own identification information.

106. The processor of claim 102, wherein the signal includes an availability information of the cache unit.

107. The processor of claim 106, wherein each of the plurality of the processing units is configured to detect the availability information thereby determining whether the cache unit is available.

108. The processor of claim 107, wherein each of the plurality of processing units is configured to wake up upon determining that the cache unit is available.

109. The processor of claim 108, wherein each of the plurality of processing units is configured to send another access request to the cache unit upon determining that the cache unit is available.

110. The processor of claim 101, wherein each of the plurality of processing units is assigned with a priority order over other processing units thereby accessing the cache unit according to the priority order.

111. A processor comprising:

a processing unit with an internal cache unit, residing internal to the processing unit, configured to execute instructions of a program; and

an external cache unit, residing external to the processing unit, configured to cooperate with the internal cache unit of the processing unit,

wherein each of the internal and external cache units is divided into an instruction area and a data area, and the data area is further subdivided into a local area and a global area.

112. The processor of claim 111, wherein the processing unit is configured to copy a portion of the global area of the internal cache into the global area of the external cache after executing a portion of the instructions.

113. The processor of claim 112, wherein the portion of the instructions is either a semaphore instructions or an unlock instructions of a program.

114. The processor of claim 111, wherein the processing unit is configured to discard a portion of the global area of the internal cache.

115. The processor of claim 111, wherein the processing unit is configured to determine whether a portion of the global area of the internal cache is accessed after executing a portion of the instructions.

116. The processor of claim 115, wherein the processing unit is configured to copy the portion of the global area of the internal cache into the global area of the external cache

after executing the portion of the instructions upon determining that the portion of the global area is accessed by the processing unit.

117. The processor of claim 115, wherein the processing unit is configured to discard the portion of the global area of the internal cache after executing the portion of the instructions upon determining that the portion of the global area is not accessed by the processing unit.

118. A method of register addressing, comprising:

obtaining a register bank address;

decoding an operation-code;

setting a cycle flag in a status register based on the decoding of the operation-code;.

accessing a register addressed by the cycle flag within the addressed register bank.

119. The method of claim 118, further comprising,

employing a set number of bits in an execution-instruction to establish the number of registers within a register bank.

120. The method of claim 118, further comprising,

employing a set number of bits in an execution-instruction to establish the number of register banks.

121. A method of setting intra-processor processing resources to sleep, comprising:

setting processing resources that issue requests to delegating processing resources to sleep until a response is provided;

setting processing resources waiting on shared and locked memory being accessed by other processing resources to sleep until the memory is unlocked.

122. A method of execution-instruction delegation between processing resources, comprising:

sharing memory between processing resources

wherein the memory itself includes instruction execution logic;

wherein the processing resources are communicatively accessible through an instruction execution router;

wherein the processing resources are on the same die;

delegating execution-instructions from originating processing resources to other processing resources.

123. The method of claim 122, wherein the method is completed within a single processing cycle.

124. The method of claim 122, wherein the other processing resource may be the originating processing resource.

125. The method of claim 122, wherein a processing resource is an integer processing unit.

126. The method of claim 122, wherein a processing resource is a mathematical processing unit.

127. The method of claim 122, wherein a processing resource is a memory management unit.

128. The method of claim 122, wherein a processing resource is a vector processing unit.

138. The method of claim 122, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

139. The method of claim 135, wherein a sleeping processing resource may be woken by a response to a request.

140. The method of claim 139, wherein the response is directed to a specific processing resource.

141. The method of claim 139, wherein the response indicates a once locked resource is unlocked.

142. The method of claim 122, wherein processing resources are communicatively disposed on a same die.

143. The method of claim 142, wherein an execution-instruction signal router is on the same die with processing resources.

144. A method of execution-instruction delegation between processing resources, comprising:

obtaining an execution instruction, wherein the execution instruction is obtained at a processing resource;

determining whether an operation-code within the execution instruction should be delegated to an other processing resource;

executing the execution instruction, if the operation-code within the execution instruction should not be delegated to an other processing resource;

routing the execution instruction to an other processing resource, if the operation-code within the execution instruction is for an other processing resource.

129. The method of claim 122, wherein a processing resource is a digital signal processing unit.

130. The method of claim 122, wherein a processing resource is a graphics processing unit.

131. The method of claim 122, wherein a processing resource is an input/output controller processing unit.

132. The method of claim 122, wherein a processing resource is an execution-instruction processing cache.

133. The method of claim 122, wherein delegation occurs through an execution-instruction signal router.

134. The method of claim 133, wherein the execution-instruction signal router is a cross-point switch.

135. The method of claim 122, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

136. The method of claim 122, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

137. The method of claim 122, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

145. The method of claim 144, wherein the method is completed within a single processing cycle.

146. The method of claim 144, wherein the routed execution instruction is a request for execution by a type of processing resource.

147. The method of claim 146, wherein the request includes an operation-code.

148. The method of claim 147, wherein the operation-code indicates a type of resource on which to execute.

149. The method of claim 146, wherein the request includes values from a status register.

150. The method of claim 146, wherein the request includes values from a priority bit status register.

151. The method of claim 146, wherein the request includes a processing resource identifier.

152. The method of claim 151, wherein the processing resource identifier is an integer processing unit number.

153. The method of claim 146, wherein the request is obtained by an execution-instruction signal router.

154. The method of claim 146, wherein the request includes an address.

155. The method of claim 144, wherein the operation-code indicates a type of resource on which to execute.

156. The method of claim 144, wherein the other processing resource may be the originating processing resource.

157. The method of claim 144, wherein a processing resource is an integer processing unit.

158. The method of claim 144, wherein a processing resource is a mathematical processing unit.

159. The method of claim 144, wherein a processing resource is a memory management unit.

160. The method of claim 144, wherein a processing resource is a vector processing unit.

161. The method of claim 144, wherein a processing resource is a digital signal processing unit.

162. The method of claim 144, wherein a processing resource is a graphics processing unit.

163. The method of claim 144, wherein a processing resource is an input/output controller processing unit.

164. The method of claim 144, wherein a processing resource is an execution-instruction processing cache.

165. The method of claim 144, wherein routing occurs through an execution-instruction signal router.

166. The method of claim 165, wherein the execution-instruction signal router is a cross-point switch.

167. The method of claim 144, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

168. The method of claim 144, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

169. The method of claim 144, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

170. The method of claim 144, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

171. The method of claim 144, wherein processing resources are communicatively disposed on a same die.

172. The method of claim 171, wherein an execution-instruction signal router is on the same die with processing resources.

173. A method of execution-instruction delegation between processing resources, comprising:

obtaining an request signal, wherein the request signal is from a requesting processing resource;

determining a priority dead-lock-avoidance value, wherein the priority dead-lock value is used to select among multiple requests with equal priority;

examining a request priority value for each submitted request;

selecting a request with a highest priority value, if more than one processing resource requests a same target resource;

providing a selected request to a target processing resource.

174. The method of claim 173, wherein the method is completed within a single processing cycle.

175. The method of claim 173, wherein the request signal is a request for execution by a type of processing resource.

176. The method of claim 175, further comprising,

routing the request to an other processing resource, if an operation-code within the request is for an other processing resource.

177. The method of claim 175, wherein the request is obtained at an execution-instruction signal router.

178. The method of claim 175, wherein the execution-instruction signal is provided by a delegating processing resource via an execution-instruction signal router.

179. The method of claim 175, wherein the request includes an operation-code.

180. The method of claim 179, wherein the operation-code indicates a type of resource on which to execute.

181. The method of claim 175, wherein the request includes values from a status register.

182. The method of claim 175, wherein the request includes values from a priority bit status register.

183. The method of claim 175, wherein the request includes a processing resource identifier.

184. The method of claim 183, wherein the processing resource identifier is an integer processing unit number.

185. The method of claim 175, wherein the request includes an address.

186. The method of claim 173, wherein the other processing resource may be the originating processing resource.

187. The method of claim 173, wherein a processing resource is an integer processing unit.

188. The method of claim 173, wherein a processing resource is a mathematical processing unit.

189. The method of claim 173, wherein a processing resource is a memory management unit.

190. The method of claim 173, wherein a processing resource is a vector processing unit.

191. The method of claim 173, wherein a processing resource is a digital signal processing unit.

192. The method of claim 173, wherein a processing resource is a graphics processing unit.

193. The method of claim 173, wherein a processing resource is an input/output controller processing unit.

194. The method of claim 173, wherein a processing resource is an execution-instruction processing cache.

195. The method of claim 173, wherein delegation occurs through an execution-instruction signal router.

196. The method of claim 195, wherein the execution-instruction signal router is a cross-point switch.

197. The method of claim 173, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

198. The method of claim 173, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

199. The method of claim 173, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

200. The method of claim 173, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

201. The method of claim 197, wherein a sleeping processing resource may be woken by a response to a request.

202. The method of claim 201, wherein the response is directed to a specific processing resource.

203. The method of claim 201, wherein the response indicates a once locked resource is unlocked.

204. The method of claim 173, wherein processing resources are communicatively disposed on a same die.

205. The method of claim 204, wherein an execution-instruction signal router is on the same die with processing resources.

206. The method of claim 173, wherein the priority dead-lock-avoidance value is iterated.

207. The method of claim 173, further comprising,

selecting a request, if not more than one processing resource requests a same target resource.

208. The method of claim 173, further comprising,

selecting the request with a highest priority value, if not more than one request has a same highest priority value;

selecting a request based on the priority dead-lock-avoidance value, if more than one request has a same highest priority value.

209. The method of claim 173, further comprising,

providing a request granted acknowledgement to the requesting processing resource whose request was selected.

210. The method of claim 173, further comprising,

clearing the operation-code in the requesting processing resource whose request was selected.

211. A method of execution-instruction delegation between processing resources, comprising:

processing a request execution-instruction, wherein the request execution-instruction includes a requesting processing resource identifier;

preparing a response into a result register, wherein the response includes the requesting processing resource identifier;

presenting the response to all processing resources.

212. The method of claim 211, wherein the method is completed within a single processing cycle.

213. The method of claim 211, wherein the request execution-instruction is a request for execution by a type of processing resource.

214. The method of claim 213, further comprising,

routing the request to an other processing resource, if an operation-code within the request is for an other processing resource.

215. The method of claim 213, wherein the request is obtained at an execution-instruction signal router.

216. The method of claim 213, wherein the execution-instruction signal is provided by a delegating processing resource via an execution-instruction signal router.

217. The method of claim 213, wherein the request includes an operation-code.

218. The method of claim 217, wherein the operation-code indicates a type of resource on which to execute.

219. The method of claim 213, wherein the request includes values from a status register.

220. The method of claim 213, wherein the request includes values from a priority bit status register.

221. The method of claim 213, wherein the request includes a processing resource identifier.

222. The method of claim 221, wherein the processing resource identifier is an integer processing unit number.

223. The method of claim 213, wherein the request includes an address.

224. The method of claim 211, wherein the other processing resource may be the originating processing resource.

225. The method of claim 211, wherein a processing resource is an integer processing unit.

226. The method of claim 211, wherein a processing resource is a mathematical processing unit.

227. The method of claim 211, wherein a processing resource is a memory management unit.

228. The method of claim 211, wherein a processing resource is a vector processing unit.

229. The method of claim 211, wherein a processing resource is a digital signal processing unit.

230. The method of claim 211, wherein a processing resource is a graphics processing unit.

231. The method of claim 211, wherein a processing resource is an input/output controller processing unit.

232. The method of claim 211, wherein a processing resource is an execution-instruction processing cache.

233. The method of claim 211, wherein delegation occurs through an execution-instruction signal router.

234. The method of claim 233, wherein the execution-instruction signal router is a cross-point switch.

235. The method of claim 211, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

236. The method of claim 211, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

237. The method of claim 211, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

238. The method of claim 211, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

239. The method of claim 235, wherein a sleeping processing resource may be woken by a response to a request.

240. The method of claim 239, wherein the response is directed to a specific processing resource.

241. The method of claim 239, wherein the response indicates a once locked resource is unlocked.

242. The method of claim 211, wherein processing resources are communicatively disposed on a same die.

243. The method of claim 242, wherein an execution-instruction signal router is on the same die with processing resources.

244. A method of execution-instruction delegation between processing resources, comprising:

obtaining a result execution-instruction from a delegate processing resource, wherein the result includes a requesting processing resource identifier;

waking the requesting processing resource, if the requesting processing resource identifier identifies the instant processing resource;

waking a processing resource, if a processing resource is waiting to be unlocked and if the obtained result includes an unlock flag and if an address in the result matches a locked address of an instant processing resource.

245. The method of claim 244, wherein the method is completed within a single processing cycle.

246. The method of claim 244, wherein the result execution-instruction is an execution-instruction signal for execution by a type of processing resource.

247. The method of claim 246, further comprising,

routing the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

248. The method of claim 246, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

249. The method of claim 246, wherein the execution-instruction signal is provided by a delegating processing resource via an execution-instruction signal router.

250. The method of claim 246, wherein the execution-instruction signal includes an operation-code.

251. The method of claim 250, wherein the operation-code indicates a type of resource on which to execute.

252. The method of claim 246, wherein the execution-instruction signal includes values from a status register.

253. The method of claim 246, wherein the execution-instruction signal includes values from a priority bit status register.

254. The method of claim 246, wherein the execution-instruction signal includes a processing resource identifier.

255. The method of claim 254, wherein the processing resource identifier is an integer processing unit number.

256. The method of claim 246, wherein the execution-instruction signal includes an address.

257. The method of claim 244, wherein an other processing resource may be an delegating processing resource.

258. The method of claim 244, wherein a processing resource is an integer processing unit.

259. The method of claim 244, wherein a processing resource is a mathematical processing unit.

260. The method of claim 244, wherein a processing resource is a memory management unit.

261. The method of claim 244, wherein a processing resource is a vector processing unit.

262. The method of claim 244, wherein a processing resource is a digital signal processing unit.

263. The method of claim 244, wherein a processing resource is a graphics processing unit.

264. The method of claim 244, wherein a processing resource is an input/output controller processing unit.

265. The method of claim 244, wherein a processing resource is an execution-instruction processing cache.

266. The method of claim 244, wherein delegation occurs through an execution-instruction signal router.

267. The method of claim 266, wherein the execution-instruction signal router is a cross-point switch.

268. The method of claim 244, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

269. The method of claim 244, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

270. The method of claim 244, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

271. The method of claim 244, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

272. The method of claim 268, wherein a sleeping processing resource may be woken by a response to a request.

273. The method of claim 272, wherein the response is directed to a specific processing resource.

274. The method of claim 272, wherein the response indicates a once locked resource is unlocked.

275. The method of claim 244, wherein processing resources are communicatively disposed on a same die.

276. The method of claim 275, wherein an execution-instruction signal router is on the same die with processing resources.

277. The method of claim 244, further comprising,

executing a next execution-instruction, if the requesting processing resource identifier identifies the instant processing resource;

executing an execution-instruction that failed to execute, if a processing resource is waiting to be unlocked and if the obtained result includes an unlock flag and if an address in the result matches a locked address of an instant processing resource.

278. A method of memory access optimization, comprising:

obtaining a storage-access request from a delegated processing resource, wherein the request includes a target memory address;

determining the target memory address from the request;

comparing the target memory address with register values, wherein register values are used to establish a data type of the target memory address for subsequent storage in an apportioned region of cache memory;

obtaining information from the target memory address;

storing the information in an apportioned region of cache memory.

279. The method of claim 278, wherein the method is completed within a single processing cycle.

280. The method of claim 278, wherein the storage-access request is an execution-instruction signal for execution by a type of processing resource.

281. The method of claim 280, further comprising,

routing the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

282. The method of claim 280, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

283. The method of claim 280, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

284. The method of claim 280, wherein the execution-instruction signal includes an operation-code.

285. The method of claim 284, wherein the operation-code indicates a type of resource on which to execute.

286. The method of claim 280, wherein the execution-instruction signal includes values from a status register.

287. The method of claim 280, wherein the execution-instruction signal includes values from a priority bit status register.

288. The method of claim 280, wherein the execution-instruction signal includes a processing resource identifier.

289. The method of claim 288, wherein the processing resource identifier is an integer processing unit number.

290. The method of claim 280, wherein the execution-instruction signal includes an address.

291. The method of claim 278, wherein an other processing resource may be an delegating processing resource.

292. The method of claim 278, wherein a processing resource is an integer processing unit.

293. The method of claim 278, wherein a processing resource is a mathematical processing unit.

294. The method of claim 278, wherein a processing resource is a memory management unit.

295. The method of claim 278, wherein a processing resource is a vector processing unit.

296. The method of claim 278, wherein a processing resource is a digital signal processing unit.

297. The method of claim 278, wherein a processing resource is a graphics processing unit.

298. The method of claim 278, wherein a processing resource is an input/output controller processing unit.

299. The method of claim 278, wherein a processing resource is an execution-instruction processing cache.

300. The method of claim 278, wherein delegation occurs through an execution-instruction signal router.

301. The method of claim 300, wherein the execution-instruction signal router is a cross-point switch.

302. The method of claim 278, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

303. The method of claim 278, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

304. The method of claim 278, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

305. The method of claim 278, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

306. The method of claim 302, wherein a sleeping processing resource may be woken by a response to a request.

307. The method of claim 306, wherein the response is directed to a specific processing resource.

308. The method of claim 306, wherein the response indicates a once locked resource is unlocked.

309. The method of claim 278, wherein processing resources are communicatively disposed on a same die.

310. The method of claim 309, wherein an execution-instruction signal router is on the same die with processing resources.

311. The method of claim 278, wherein the storage-access request is obtained at a processing resource.

312. The method of claim 278, wherein information at the target memory address is designated to be local data if the target memory address is within an address range established by stack base and end register values.

313. The method of claim 278, wherein information at the target memory address is designated to be global data if the target memory address is outside an address range established by stack base and end register values.

314. The method of claim 278, wherein an apportionment is designated for each type of data for optimized access.

315. The method of claim 278, wherein the apportioned region is a hash region.

316. The method of claim 315, wherein a hash region is designated for local data.

317. The method of claim 315, wherein a hash region is designated for global data.

318. The method of claim 315, wherein the apportionment for a local data region is greater than a region for global data.

319. A method of reduced size execution of execution-instructions, comprising:

obtaining an execution-instruction at a processing resource;

appending a literal constant in a literal register, if a literal flag is set by examining a special register and if there is literal prefix in the execution-instruction;

executing an execution-instruction with a constant in a literal register, if a literal flag is set by examining a special register and if there is no literal prefix in the execution-instruction;

setting a literal constant flag in a status register and placing a literal constant in a literal register, if a literal flag is not set by examining a special register and if there is literal prefix in the execution-instruction;

executing an execution-instruction, if a literal flag is not set by examining a special register and if there is no literal prefix in the execution-instruction

320. The method of claim 319, wherein the method is completed within a single processing cycle.

321. The method of claim 319, wherein the execution-instruction is an execution-instruction signal for execution by a type of processing resource.

322. The method of claim 321, further comprising,

routing the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

323. The method of claim 321, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

324. The method of claim 321, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

325. The method of claim 321, wherein the execution-instruction signal includes an operation-code.

326. The method of claim 325, wherein the operation-code indicates a type of resource on which to execute.

327. The method of claim 321, wherein the execution-instruction signal includes values from a status register.

328. The method of claim 321, wherein the execution-instruction signal includes values from a priority bit status register.

329. The method of claim 321, wherein the execution-instruction signal includes a processing resource identifier.

330. The method of claim 329, wherein the processing resource identifier is an integer processing unit number.

331. The method of claim 321, wherein the execution-instruction signal includes an address.

332. The method of claim 319, wherein an other processing resource may be an delegating processing resource.

333. The method of claim 319, wherein a processing resource is an integer processing unit.

334. The method of claim 319, wherein a processing resource is a mathematical processing unit.

335. The method of claim 319, wherein a processing resource is a memory management unit.

336. The method of claim 319, wherein a processing resource is a vector processing unit.

337. The method of claim 319, wherein a processing resource is a digital signal processing unit.

338. The method of claim 319, wherein a processing resource is a graphics processing unit.

339. The method of claim 319, wherein a processing resource is an input/output controller processing unit.

340. The method of claim 319, wherein a processing resource is an execution-instruction processing cache.

341. The method of claim 319, wherein delegation occurs through an execution-instruction signal router.

342. The method of claim 341, wherein the execution-instruction signal router is a cross-point switch.

343. The method of claim 319, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

344. The method of claim 319, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

345. The method of claim 319, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

346. The method of claim 319, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

347. The method of claim 343, wherein a sleeping processing resource may be woken by a response to a request.

348. The method of claim 347, wherein the response is directed to a specific processing resource.

349. The method of claim 347, wherein the response indicates a once locked resource is unlocked.

350. The method of claim 319, wherein processing resources are communicatively disposed on a same die.

351. The method of claim 350, wherein an execution-instruction signal router is on the same die with processing resources.

352. The method of claim 319, wherein the literal constant is appended by employing a shift.

353. The method of claim 319, wherein the processing resource appends the literal constant.

354. The method of claim 319, wherein the execution-instruction is executed as an extended execution of an operation-code.

355. The method of claim 354, wherein a processing resource executes the execution-instruction.

356. The method of claim 319, wherein the status register is in the processing resource.

357. The method of claim 319, wherein the literal register is in the processing resource.

358. The method of claim 319, wherein the execution-instruction is executed as a non-extended execution of an operation-code.

359. The method of claim 358, wherein a processing resource executes the execution-instruction.

360. A method of binding instructions, comprising:

storing binding names at odd addresses;

generating a binding name interrupt, if a processing resource attempts to load an instruction register with an odd address value;

providing an operating system with a binding name interrupt;

performing a look-up of the odd address value that generated the binding name interrupt in a binding name table, which was established by linker;

replacing the odd address value in an instruction register with an even address found in the binding name table.

361. The method of claim 360, wherein the interrupt is an execution-instruction signal for execution by a type of processing resource.

362. The method of claim 361, further comprising,

routing the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

363. The method of claim 361, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

364. The method of claim 361, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

365. The method of claim 361, wherein the execution-instruction signal includes an operation-code.

366. The method of claim 365, wherein the operation-code indicates a type of resource on which to execute.

367. The method of claim 361, wherein the execution-instruction signal includes values from a status register.

368. The method of claim 361, wherein the execution-instruction signal includes values from a priority bit status register.

369. The method of claim 361, wherein the execution-instruction signal includes a processing resource identifier.

370. The method of claim 369, wherein the processing resource identifier is an integer processing unit number.

371. The method of claim 361, wherein the execution-instruction signal includes an address.

372. The method of claim 360, wherein an other processing resource may be an delegating processing resource.

373. The method of claim 360, wherein a processing resource is an integer processing unit.

374. The method of claim 360, wherein a processing resource is a mathematical processing unit.

375. The method of claim 360, wherein a processing resource is a memory management unit.

376. The method of claim 360, wherein a processing resource is a vector processing unit.

377. The method of claim 360, wherein a processing resource is a digital signal processing unit.

378. The method of claim 360, wherein a processing resource is a graphics processing unit.

379. The method of claim 360, wherein a processing resource is an input/output controller processing unit.

380. The method of claim 360, wherein a processing resource is an execution-instruction processing cache.

381. The method of claim 360, wherein delegation occurs through an execution-instruction signal router.

382. The method of claim 381, wherein the execution-instruction signal router is a cross-point switch.

383. The method of claim 360, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

384. The method of claim 360, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

385. The method of claim 360, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

386. The method of claim 360, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

387. The method of claim 383, wherein a sleeping processing resource may be woken by a response to a request.

388. The method of claim 387, wherein the response is directed to a specific processing resource.

389. The method of claim 387, wherein the response indicates a once locked resource is unlocked.

390. The method of claim 360, wherein processing resources are communicatively disposed on a same die.

391. The method of claim 390, wherein an execution-instruction signal router is on the same die with processing resources.

392. The method of claim 360, wherein the binding names are stored by a linker.

393. The method of claim 360, further comprising,

replacing the odd address value stored in an instruction stack with an instruction pointer at the even address, if late binding is being employed.

394. The method of claim 360, further comprising,

replacing the odd address value stored in an instruction stack with an instruction pointer at the even address and replacing the odd address value in the binding name table with the even address, if dynamic binding is being employed.

395. A method of addressing registers, comprising:

determining if an execution-instruction operation-code provides additional register addressing information;

setting cycle flags in a status register with the additional register addressing information;

examining an execution-instruction for a register address;

addressing a register specified by the register address and cycle flags.

396. The method of claim 395, wherein the method is completed within a single processing cycle.

397. The method of claim 395, wherein the execution-instruction is an execution-instruction signal for execution by a type of processing resource.

398. The method of claim 397, further comprising,

routing the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

399. The method of claim 397, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

400. The method of claim 397, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

401. The method of claim 397, wherein the execution-instruction signal includes an operation-code.

402. The method of claim 401, wherein the operation-code indicates a type of resource on which to execute.

403. The method of claim 397, wherein the execution-instruction signal includes values from a status register.

404. The method of claim 397, wherein the execution-instruction signal includes values from a priority bit status register.

405. The method of claim 397, wherein the execution-instruction signal includes a processing resource identifier.

406. The method of claim 405, wherein the processing resource identifier is an integer processing unit number.

407. The method of claim 397, wherein the execution-instruction signal includes an address.

408. The method of claim 395, wherein an other processing resource may be an delegating processing resource.

409. The method of claim 395, wherein a processing resource is an integer processing unit.

410. The method of claim 395, wherein a processing resource is a mathematical processing unit.

411. The method of claim 395, wherein a processing resource is a memory management unit.

412. The method of claim 395, wherein a processing resource is a vector processing unit.

413. The method of claim 395, wherein a processing resource is a digital signal processing unit.

414. The method of claim 395, wherein a processing resource is a graphics processing unit.

415. The method of claim 395, wherein a processing resource is an input/output controller processing unit.

416. The method of claim 395, wherein a processing resource is an execution-instruction processing cache.

417. The method of claim 395, wherein delegation occurs through an execution-instruction signal router.

418. The method of claim 417, wherein the execution-instruction signal router is a cross-point switch.

419. The method of claim 395, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

420. The method of claim 395, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

421. The method of claim 395, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

422. The method of claim 395, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

423. The method of claim 419, wherein a sleeping processing resource may be woken by a response to a request.

424. The method of claim 423, wherein the response is directed to a specific processing resource.

425. The method of claim 423, wherein the response indicates a once locked resource is unlocked.

426. The method of claim 395, wherein processing resources are communicatively disposed on a same die.

427. The method of claim 426, wherein an execution-instruction signal router is on the same die with processing resources.

428. The method of claim 395, wherein specific operation-codes address extended register sets without requiring dedicated bits set aside for register addressing in the execution-instruction.

429. The method of claim 395, wherein the register address comprises 3 bits of the execution-instruction.

430. The method of claim 395, wherein a value of a cycle flag indicates that an operation-code is a multi-cycle operation.

431. A method of sharing memory, comprising:

obtaining a request execution-instruction from a delegate processing resource at an instruction processing cache, wherein the request includes a target memory address;

determining if the target address is locked based on a lock variable at the target memory address;

determining what type of lock is provided in the request;

requesting that a requesting processing resource sleep until it is unlocked, if the target memory address is locked.

432. The method of claim 431, wherein the method is completed within a single processing cycle.

433. The method of claim 431, wherein the request execution-instruction is an execution-instruction signal for execution by a type of processing resource.

434. The method of claim 433, further comprising,

routing the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

435. The method of claim 433, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

436. The method of claim 433, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

437. The method of claim 433, wherein the execution-instruction signal includes an operation-code.

438. The method of claim 437, wherein the operation-code indicates a type of resource on which to execute.

439. The method of claim 433, wherein the execution-instruction signal includes values from a status register.

440. The method of claim 433, wherein the execution-instruction signal includes values from a priority bit status register.

441. The method of claim 433, wherein the execution-instruction signal includes a processing resource identifier.

442. The method of claim 441, wherein the processing resource identifier is an integer processing unit number.

443. The method of claim 433, wherein the execution-instruction signal includes an address.

444. The method of claim 431, wherein an other processing resource may be an delegating processing resource.

445. The method of claim 431, wherein a processing resource is an integer processing unit.

446. The method of claim 431, wherein a processing resource is a mathematical processing unit.

447. The method of claim 431, wherein a processing resource is a memory management unit.

448. The method of claim 431, wherein a processing resource is a vector processing unit.

449. The method of claim 431, wherein a processing resource is a digital signal processing unit.

450. The method of claim 431, wherein a processing resource is a graphics processing unit.

451. The method of claim 431, wherein a processing resource is an input/output controller processing unit.

452. The method of claim 431, wherein a processing resource is an execution-instruction processing cache.

453. The method of claim 431, wherein delegation occurs through an execution-instruction signal router.

454. The method of claim 453, wherein the execution-instruction signal router is a cross-point switch.

455. The method of claim 431, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

456. The method of claim 431, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

457. The method of claim 431, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

458. The method of claim 431, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

459. The method of claim 455, wherein a sleeping processing resource may be woken by a response to a request.

460. The method of claim 459, wherein the response is directed to a specific processing resource.

461. The method of claim 459, wherein the response indicates a once locked resource is unlocked.

462. The method of claim 431, wherein processing resources are communicatively disposed on a same die.

463. The method of claim 462, wherein an execution-instruction signal router is on the same die with processing resources.

464. The method of claim 431, wherein the determination is made by the instruction processing cache's logic facilities.

465. The method of claim 431, wherein the requesting processing resource is unlocked when an unlock instruction is received that specifies the same target memory address.

466. The method of claim 431, wherein the target memory address is locked for read and write operations.

467. The method of claim 431, wherein the target memory address is locked for read only operations and the request specifies a read and write operation lock.

468. A method of sharing memory, comprising:

obtaining a response to an execution-instruction from a delegate processing resource, wherein the response includes a target address at a processing resource;

determining if a target address is locked based on a lock variable at the target memory address;

determining value types by using a hash function;

updating value types in a primary cache memory of a processing resource to a secondary cache memory for each value type, if each value type in a primary cache memory of a processing resource has not been updated to a secondary cache memory;

469. The method of claim 468, wherein the method is completed within a single processing cycle. \

470. The method of claim 468, wherein the response is an execution-instruction signal for execution by a type of processing resource.

471. The method of claim 470, further comprising,

routing the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

472. The method of claim 470, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

473. The method of claim 470, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

474. The method of claim 470, wherein the execution-instruction signal includes an operation-code.

475. The method of claim 474, wherein the operation-code indicates a type of resource on which to execute.

476. The method of claim 470, wherein the execution-instruction signal includes values from a status register.

477. The method of claim 470, wherein the execution-instruction signal includes values from a priority bit status register.

478. The method of claim 470, wherein the execution-instruction signal includes a processing resource identifier.

479. The method of claim 478, wherein the processing resource identifier is an integer processing unit number.

480. The method of claim 470, wherein the execution-instruction signal includes an address.

481. The method of claim 468, wherein an other processing resource may be an delegating processing resource.

482. The method of claim 468, wherein a processing resource is an integer processing unit.

483. The method of claim 468, wherein a processing resource is a mathematical processing unit.

484. The method of claim 468, wherein a processing resource is a memory management unit.

485. The method of claim 468, wherein a processing resource is a vector processing unit.

486. The method of claim 468, wherein a processing resource is a digital signal processing unit.

487. The method of claim 468, wherein a processing resource is a graphics processing unit.

488. The method of claim 468, wherein a processing resource is an input/output controller processing unit.

489. The method of claim 468, wherein a processing resource is an execution-instruction processing cache.

490. The method of claim 468, wherein delegation occurs through an execution-instruction signal router.

491. The method of claim 490, wherein the execution-instruction signal router is a cross-point switch.

492. The method of claim 468, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

493. The method of claim 468, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

494. The method of claim 468, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

495. The method of claim 468, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

496. The method of claim 492, wherein a sleeping processing resource may be woken by a response to a request.

497. The method of claim 496, wherein the response is directed to a specific processing resource.

498. The method of claim 496, wherein the response indicates a once locked resource is unlocked.

499. The method of claim 468, wherein processing resources are communicatively disposed on a same die.

500. The method of claim 499, wherein an execution-instruction signal router is on the same die with processing resources.

501. The method of claim 468, wherein the response includes a target processing resource identifier.

502. The method of claim 468, wherein the determination is made by a processing resource.

503. The method of claim 468, wherein the value type is a global value.

504. The method of claim 468, wherein the secondary cache memory is a Level 2 cache memory.

505. The method of claim 468, wherein a primary cache memory is a Level 1 cache memory.

506. The method of claim 468, further comprising,

marking a cache line of the updated value type as free. 507. The method of claim 468, further comprising,

updating lock variables in secondary cache memory for each updated value type.

507. The method of claim 507, wherein the lock variable updating is performed by the secondary cache memory, which is a processing cache memory.

508. A method of sharing memory, comprising:

obtaining a storage request including an event variable address and execution-instruction for a processing resource to sleep;

determining if an event variable is set;

providing a reply to a processing resource bus with the event variable address and event value, which will wake an appropriate processing resource from sleep, if the event value is set not to sleep;

setting a sleep until event value for the event variable, if the event value is set to sleep.

509. A method of sharing memory, comprising:

obtaining a storage request including an event variable address and execution instruction for a processing resource to sleep;

determining if an event variable is set;

providing a reply to a processing resource bus with the event variable address and event value, which will wake an appropriate processing resource from sleep, if the event value is set not to sleep;

setting a sleep until event value for the event variable, if the event value is set to sleep.

510. The method of claim 509, wherein the method is completed within a single processing cycle.

511. The method of claim 509, wherein the storage request is an execution-instruction signal for execution by a type of processing resource.

512. The method of claim 511, further comprising,

routing the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

513. The method of claim 511, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

514. The method of claim 511, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

515. The method of claim 511, wherein the execution-instruction signal includes an operation-code.

516. The method of claim 515, wherein the operation-code indicates a type of resource on which to execute.

517. The method of claim 511, wherein the execution-instruction signal includes values from a status register.

518. The method of claim 511, wherein the execution-instruction signal includes values from a priority bit status register.

519. The method of claim 511, wherein the execution-instruction signal includes a processing resource identifier.

520. The method of claim 519, wherein the processing resource identifier is an integer processing unit number.

521. The method of claim 511, wherein the execution-instruction signal includes an address.

522. The method of claim 509, wherein an other processing resource may be an delegating processing resource.

523. The method of claim 509, wherein a processing resource is an integer processing unit.

524. The method of claim 509, wherein a processing resource is a mathematical processing unit.

525. The method of claim 509, wherein a processing resource is a memory management unit.

526. The method of claim 509, wherein a processing resource is a vector processing unit.

527. The method of claim 509, wherein a processing resource is a digital signal processing unit.

528. The method of claim 509, wherein a processing resource is a graphics processing unit.

529. The method of claim 509, wherein a processing resource is an input/output controller processing unit.

530. The method of claim 509, wherein a processing resource is an execution-instruction processing cache.

531. The method of claim 509, wherein delegation occurs through an execution-instruction signal router.

532. The method of claim 531, wherein the execution-instruction signal router is a cross-point switch.

533. The method of claim 509, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

534. The method of claim 509, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

535. The method of claim 509, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

536. The method of claim 509, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

537. The method of claim 533, wherein a sleeping processing resource may be woken by a response to a request.

538. The method of claim 537, wherein the response is directed to a specific processing resource.

539. The method of claim 537, wherein the response indicates a once locked resource is unlocked.

540. The method of claim 509, wherein processing resources are communicatively disposed on a same die.

541. The method of claim 540, wherein an execution-instruction signal router is on the same die with processing resources.

542. The method of claim 509, wherein the determination is made at a processing cache.

543. The method of claim 509, wherein the event variable is set at the processing cache.

544. The method of claim 509, wherein the event variable is set in a register.

545. The method of claim 509, wherein the event variable is set in a target address in a processing cache.

546. A method of sharing memory, comprising:

obtaining a storage request including an event address at a processing cache;

combining a parameter with an event variable through in a processing cache;

providing a reply to a processing resource bus from the combination with the event address, which will wake an appropriate processing resource from sleep that is waiting on an event, if the event value is set not to sleep.

547. The method of claim 546, wherein the method is completed within a single processing cycle.

548. The method of claim 546, wherein the storage request is an execution-instruction signal for execution by a type of processing resource.

549. The method of claim 548, further comprising,

routing the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

550. The method of claim 548, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

551. The method of claim 548, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

552. The method of claim 548, wherein the execution-instruction signal includes an operation-code.

553. The method of claim 552, wherein the operation-code indicates a type of resource on which to execute.

554. The method of claim 548, wherein the execution-instruction signal includes values from a status register.

555. The method of claim 548, wherein the execution-instruction signal includes values from a priority bit status register.

556. The method of claim 548, wherein the execution-instruction signal includes a processing resource identifier.

557. The method of claim 556, wherein the processing resource identifier is an integer processing unit number.

558. The method of claim 548, wherein the execution-instruction signal includes an address.

559. The method of claim 546, wherein an other processing resource may be an delegating processing resource.

560. The method of claim 546, wherein a processing resource is an integer processing unit.

561. The method of claim 546, wherein a processing resource is a mathematical processing unit.

562. The method of claim 546, wherein a processing resource is a memory management unit.

563. The method of claim 546, wherein a processing resource is a vector processing unit.

564. The method of claim 546, wherein a processing resource is a digital signal processing unit.

565. The method of claim 546, wherein a processing resource is a graphics processing unit.

566. The method of claim 546, wherein a processing resource is an input/output controller processing unit.

567. The method of claim 546, wherein a processing resource is an execution-instruction processing cache.

568. The method of claim 546, wherein delegation occurs through an execution-instruction signal router.

569. The method of claim 568, wherein the execution-instruction signal router is a cross-point switch.

570. The method of claim 546, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

571. The method of claim 546, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

572. The method of claim 546, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

573. The method of claim 546, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

574. The method of claim 570, wherein a sleeping processing resource may be woken by a response to a request.

575. The method of claim 574, wherein the response is directed to a specific processing resource.

576. The method of claim 574, wherein the response indicates a once locked resource is unlocked.

577. The method of claim 546, wherein processing resources are communicatively disposed on a same die.

578. The method of claim 577, wherein an execution-instruction signal router is on the same die with processing resources.

579. The method of claim 546, wherein the parameter is a bit-mask.

580. The method of claim 546, wherein the combination is a logical-OR.

581. A method of sharing memory, comprising:

obtaining a storage request including a semaphore address at a processing cache;

setting a semaphore variable in a processing cache at the semaphore address;

providing an operating system trap-call to wait on a processing queue, if the storage request includes a wait-on-semaphore instruction;

providing an operating system trap-call to signal on a processing queue, if the storage request includes a signal-on-semaphore instruction;

582. The method of claim 581, wherein the method is completed within a single processing cycle.

583. The method of claim 581, wherein the storage request is an execution-instruction signal for execution by a type of processing resource.

584. The method of claim 583, further comprising,

routing the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

585. The method of claim 583, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

586. The method of claim 583, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

587. The method of claim 583, wherein the execution-instruction signal includes an operation-code.

588. The method of claim 587, wherein the operation-code indicates a type of resource on which to execute.

589. The method of claim 583, wherein the execution-instruction signal includes values from a status register.

590. The method of claim 583, wherein the execution-instruction signal includes values from a priority bit status register.

591. The method of claim 583, wherein the execution-instruction signal includes a processing resource identifier.

592. The method of claim 591, wherein the processing resource identifier is an integer processing unit number.

593. The method of claim 583, wherein the execution-instruction signal includes an address.

594. The method of claim 581, wherein an other processing resource may be an delegating processing resource.

595. The method of claim 581, wherein a processing resource is an integer processing unit.

596. The method of claim 581, wherein a processing resource is a mathematical processing unit.

597. The method of claim 581, wherein a processing resource is a memory management unit.

598. The method of claim 581, wherein a processing resource is a vector processing unit.

599. The method of claim 581, wherein a processing resource is a digital signal processing unit.

600. The method of claim 581, wherein a processing resource is a graphics processing unit.

601. The method of claim 581, wherein a processing resource is an input/output controller processing unit.

602. The method of claim 581, wherein a processing resource is an execution-instruction processing cache.

603. The method of claim 581, wherein delegation occurs through an execution-instruction signal router.

604. The method of claim 603, wherein the execution-instruction signal router is a cross-point switch.

605. The method of claim 581, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

606. The method of claim 581, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

607. The method of claim 581, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

608. The method of claim 581, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

609. The method of claim 605, wherein a sleeping processing resource may be woken by a response to a request.

610. The method of claim 609, wherein the response is directed to a specific processing resource.

611. The method of claim 609, wherein the response indicates a once locked resource is unlocked.

612. The method of claim 581, wherein processing resources are communicatively disposed on a same die.

613. The method of claim 612, wherein an execution-instruction signal router is on the same die with processing resources.

614. The method of claim 581, wherein the trap-call causes rescheduling so that other processes can run on a particular processing resource.

615. The method of claim 581, wherein other processes include threads.

616. The method of claim 581, wherein other processes include threads.

617. A system of register addressing, comprising:

means to obtain a register bank address;

means to decode an operation-code;

means to set a cycle flag in a status register based on the decoding of the operation-code;.

means to access a register addressed by the cycle flag within the addressed register bank.

618. The system of claim 617, further comprising,

means to employ a set number of bits in an execution-instruction to establish the number of registers within a register bank.

619. The system of claim 617, further comprising,

means to employ a set number of bits in an execution-instruction to establish the number of register banks.

620. A system of setting intra-processor processing resources to sleep, comprising:

means to set processing resources that issue requests to delegating processing resources to sleep until a response is provided;

means to set processing resources waiting on shared and locked memory being accessed by other processing resources to sleep until the memory is unlocked.

621. A system of execution-instruction delegation between processing resources, comprising:

means to share memory between processing resources

wherein the memory itself includes instruction execution logic;

wherein the processing resources are communicatively accessible through an instruction execution router;

wherein the processing resources are on the same die;

means to delegate execution-instructions from originating processing resources to other processing resources.

622. The system of claim 621, wherein the system is completed within a single processing cycle.

623. The system of claim 621, wherein the other processing resource may be the originating processing resource.

624. The system of claim 621, wherein a processing resource is an integer processing unit.

625. The system of claim 621, wherein a processing resource is a mathematical processing unit.

626. The system of claim 621, wherein a processing resource is a memory management unit.

627. The system of claim 621, wherein a processing resource is a vector processing unit.

628. The system of claim 621, wherein a processing resource is a digital signal processing unit.

629. The system of claim 621, wherein a processing resource is a graphics processing unit.

630. The system of claim 621, wherein a processing resource is an input/output controller processing unit.

631. The system of claim 621, wherein a processing resource is an execution-instruction processing cache.

632. The system of claim 621, wherein delegation occurs through an execution-instruction signal router.

633. The system of claim 632, wherein the execution-instruction signal router is a cross-point switch.

634. The system of claim 621, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

635. The system of claim 621, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

636. The system of claim 621, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

637. The system of claim 621, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

638. The system of claim 634, wherein a sleeping processing resource may be woken by a response to a request.

639. The system of claim 638, wherein the response is directed to a specific processing resource.

640. The system of claim 638, wherein the response indicates a once locked resource is unlocked.

641. The system of claim 621, wherein processing resources are communicatively disposed on a same die.

642. The system of claim 641, wherein an execution-instruction signal router is on the same die with processing resources.

643. A system of execution-instruction delegation between processing resources, comprising:

means to obtain an execution instruction, wherein the execution instruction is obtained at a processing resource;

means to determine whether an operation-code within the execution instruction should be delegated to an other processing resource;

means to execute the execution instruction, if the operation-code within the execution instruction should not be delegated to an other processing resource;

means to route the execution instruction to an other processing resource, if the operation-code within the execution instruction is for an other processing resource.

644. The system of claim 643, wherein the system is completed within a single processing cycle.

645. The system of claim 643, wherein the routed execution instruction is a request for execution by a type of processing resource.

646. The system of claim 645, wherein the request includes an operation-code.

647. The system of claim 646, wherein the operation-code indicates a type of resource on which to execute.

648. The system of claim 645, wherein the request includes values from a status register.

649. The system of claim 645, wherein the request includes values from a priority bit status register.

650. The system of claim 645, wherein the request includes a processing resource identifier.

651. The system of claim 650, wherein the processing resource identifier is an integer processing unit number.

652. The system of claim 645, wherein the request is obtained by an execution-instruction signal router.

653. The system of claim 645, wherein the request includes an address.

654. The system of claim 643, wherein the operation-code indicates a type of resource on which to execute.

655. The system of claim 643, wherein the other processing resource may be the originating processing resource.

656. The system of claim 643, wherein a processing resource is an integer processing unit.

657. The system of claim 643, wherein a processing resource is a mathematical processing unit.

658. The system of claim 643, wherein a processing resource is a memory management unit.

659. The system of claim 643, wherein a processing resource is a vector processing unit.

660. The system of claim 643, wherein a processing resource is a digital signal processing unit.

661. The system of claim 643, wherein a processing resource is a graphics processing unit.

662. The system of claim 643, wherein a processing resource is an input/output controller processing unit.

663. The system of claim 643, wherein a processing resource is an execution-instruction processing cache.

664. The system of claim 643, wherein routing occurs through an execution-instruction signal router.

665. The system of claim 664, wherein the execution-instruction signal router is a cross-point switch.

666. The system of claim 643, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

667. The system of claim 643, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

668. The system of claim 643, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

669. The system of claim 643, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

670. The system of claim 643, wherein processing resources are communicatively disposed on a same die.

671. The system of claim 670, wherein an execution-instruction signal router is on the same die with processing resources.

672. A system of execution-instruction delegation between processing resources, comprising:

means to obtain an request signal, wherein the request signal is from a requesting processing resource;

means to determine a priority dead-lock-avoidance value, wherein the priority dead-lock value is used to select among multiple requests with equal priority;

means to examine a request priority value for each submitted request;

means to select a request with a highest priority value, if more than one processing resource requests a same target resource;

means to provide a selected request to a target processing resource.

673. The system of claim 672, wherein the system is completed within a single processing cycle.

674. The system of claim 672, wherein the request signal is a request for execution by a type of processing resource.

675. The system of claim 674, further comprising,

means to route the request to an other processing resource, if an operation-code within the request is for an other processing resource.

676. The system of claim 674, wherein the request is obtained at an execution-instruction signal router.

677. The system of claim 674, wherein the execution-instruction signal is provided by a delegating processing resource via an execution-instruction signal router.

678. The system of claim 674, wherein the request includes an operation-code.

679. The system of claim 678, wherein the operation-code indicates a type of resource on which to execute.

680. The system of claim 674, wherein the request includes values from a status register.

681. The system of claim 674, wherein the request includes values from a priority bit status register.

682. The system of claim 674, wherein the request includes a processing resource identifier.

683. The system of claim 682, wherein the processing resource identifier is an integer processing unit number.

684. The system of claim 674, wherein the request includes an address.

685. The system of claim 672, wherein the other processing resource may be the originating processing resource.

686. The system of claim 672, wherein a processing resource is an integer processing unit.

687. The system of claim 672, wherein a processing resource is a mathematical processing unit.

688. The system of claim 672, wherein a processing resource is a memory management unit.

689. The system of claim 672, wherein a processing resource is a vector processing unit.

690. The system of claim 672, wherein a processing resource is a digital signal processing unit.

691. The system of claim 672, wherein a processing resource is a graphics processing unit.

692. The system of claim 672, wherein a processing resource is an input/output controller processing unit.

693. The system of claim 672, wherein a processing resource is an execution-instruction processing cache.

694. The system of claim 672, wherein delegation occurs through an execution-instruction signal router.

695. The system of claim 694, wherein the execution-instruction signal router is a cross-point switch.

696. The system of claim 672, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

697. The system of claim 672, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

698. The system of claim 672, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

699. The system of claim 672, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

700. The system of claim 696, wherein a sleeping processing resource may be woken by a response to a request.

701. The system of claim 700, wherein the response is directed to a specific processing resource.

702. The system of claim 700, wherein the response indicates a once locked resource is unlocked.

703. The system of claim 672, wherein processing resources are communicatively disposed on a same die.

704. The system of claim 703, wherein an execution-instruction signal router is on the same die with processing resources.

705. The system of claim 672, wherein the priority dead-lock-avoidance value is iterated.

706. The system of claim 672, further comprising,

means to select a request, if not more than one processing resource requests a same target resource.

707. The system of claim 672, further comprising,

means to select the request with a highest priority value, if not more than one request has a same highest priority value;

means to select a request based on the priority dead-lock-avoidance value, if more than one request has a same highest priority value.

708. The system of claim 672, further comprising,

means to provide a request granted acknowledgement to the requesting processing resource whose request was selected.

709. The system of claim 672, further comprising,

means to clear the operation-code in the requesting processing resource whose request was selected.

710. A system of execution-instruction delegation between processing resources, comprising:

means to process a request execution-instruction, wherein the request execution-instruction includes a requesting processing resource identifier;

means to prepare a response into a result register, wherein the response includes the requesting processing resource identifier;

means to present the response to all processing resources.

711. The system of claim 710, wherein the system is completed within a single processing cycle.

712. The system of claim 710, wherein the request execution-instruction is a request for execution by a type of processing resource.

713. The system of claim 712, further comprising,

means to route the request to an other processing resource, if an operation-code within the request is for an other processing resource.

714. The system of claim 712, wherein the request is obtained at an execution-instruction signal router.

715. The system of claim 712, wherein the execution-instruction signal is provided by a delegating processing resource via an execution-instruction signal router.

716. The system of claim 712, wherein the request includes an operation-code.

717. The system of claim 716, wherein the operation-code indicates a type of resource on which to execute.

718. The system of claim 712, wherein the request includes values from a status register.

719. The system of claim 712, wherein the request includes values from a priority bit status register.

720. The system of claim 712, wherein the request includes a processing resource identifier.

721. The system of claim 720, wherein the processing resource identifier is an integer processing unit number.

722. The system of claim 712, wherein the request includes an address.

723. The system of claim 710, wherein the other processing resource may be the originating processing resource.

724. The system of claim 710, wherein a processing resource is an integer processing unit.

725. The system of claim 710, wherein a processing resource is a mathematical processing unit.

726. The system of claim 710, wherein a processing resource is a memory management unit.

727. The system of claim 710, wherein a processing resource is a vector processing unit.

728. The system of claim 710, wherein a processing resource is a digital signal processing unit.

729. The system of claim 710, wherein a processing resource is a graphics processing unit.

730. The system of claim 710, wherein a processing resource is an input/output controller processing unit.

731. The system of claim 710, wherein a processing resource is an execution-instruction processing cache.

732. The system of claim 710, wherein delegation occurs through an execution-instruction signal router.

733. The system of claim 732, wherein the execution-instruction signal router is a cross-point switch.

734. The system of claim 710, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

735. The system of claim 710, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

736. The system of claim 710, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

737. The system of claim 710, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

738. The system of claim 734, wherein a sleeping processing resource may be woken by a response to a request.

739. The system of claim 738, wherein the response is directed to a specific processing resource.

740. The system of claim 738, wherein the response indicates a once locked resource is unlocked.

741. The system of claim 710, wherein processing resources are communicatively disposed on a same die.

742. The system of claim 741, wherein an execution-instruction signal router is on the same die with processing resources.

743. A system of execution-instruction delegation between processing resources, comprising:

means to obtain a result execution-instruction from a delegate processing resource, wherein the result includes a requesting processing resource identifier;

means to wake the requesting processing resource, if the requesting processing resource identifier identifies the instant processing resource;

means to wake a processing resource, if a processing resource is waiting to be unlocked and if the obtained result includes an unlock flag and if an address in the result matches a locked address of an instant processing resource.

744. The system of claim 743, wherein the system is completed within a single processing cycle.

745. The system of claim 743, wherein the result execution-instruction is an execution-instruction signal for execution by a type of processing resource.

746. The system of claim 745, further comprising,

means to route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

747. The system of claim 745, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

748. The system of claim 745, wherein the execution-instruction signal is provided by a delegating processing resource via an execution-instruction signal router.

749. The system of claim 745, wherein the execution-instruction signal includes an operation-code.

750. The system of claim 749, wherein the operation-code indicates a type of resource on which to execute.

751. The system of claim 745, wherein the execution-instruction signal includes values from a status register.

752. The system of claim 745, wherein the execution-instruction signal includes values from a priority bit status register.

753. The system of claim 745, wherein the execution-instruction signal includes a processing resource identifier.

754. The system of claim 753, wherein the processing resource identifier is an integer processing unit number.

755. The system of claim 745, wherein the execution-instruction signal includes an address.

756. The system of claim 743, wherein an other processing resource may be an delegating processing resource.

757. The system of claim 743, wherein a processing resource is an integer processing unit.

758. The system of claim 743, wherein a processing resource is a mathematical processing unit.

759. The system of claim 743, wherein a processing resource is a memory management unit.

760. The system of claim 743, wherein a processing resource is a vector processing unit.

761. The system of claim 743, wherein a processing resource is a digital signal processing unit.

762. The system of claim 743, wherein a processing resource is a graphics processing unit.

763. The system of claim 743, wherein a processing resource is an input/output controller processing unit.

764. The system of claim 743, wherein a processing resource is an execution-instruction processing cache.

765. The system of claim 743, wherein delegation occurs through an execution-instruction signal router.

766. The system of claim 765, wherein the execution-instruction signal router is a cross-point switch.

767. The system of claim 743, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

768. The system of claim 743, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

769. The system of claim 743, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

770. The system of claim 743, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

771. The system of claim 767, wherein a sleeping processing resource may be woken by a response to a request.

772. The system of claim 771, wherein the response is directed to a specific processing resource.

773. The system of claim 771, wherein the response indicates a once locked resource is unlocked.

774. The system of claim 743, wherein processing resources are communicatively disposed on a same die.

775. The system of claim 774, wherein an execution-instruction signal router is on the same die with processing resources.

776. The system of claim 743, further comprising,

means to execute a next execution-instruction, if the requesting processing resource identifier identifies the instant processing resource;

means to execute an execution-instruction that failed to execute, if a processing resource is waiting to be unlocked and if the obtained result includes an unlock flag and if an address in the result matches a locked address of an instant processing resource.

777. A system of memory access optimization, comprising:

means to obtain a storage-access request from a delegated processing resource, wherein the request includes a target memory address;

means to determine the target memory address from the request;

means to compare the target memory address with register values, wherein register values are used to establish a data type of the target memory address for subsequent storage in an apportioned region of cache memory;

means to obtain information from the target memory address;

means to store the information in an apportioned region of cache memory.

778. The system of claim 777, wherein the system is completed within a single processing cycle.

779. The system of claim 777, wherein the storage-access request is an execution-instruction signal for execution by a type of processing resource.

780. The system of claim 779, further comprising,

means to route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

781. The system of claim 779, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

782. The system of claim 779, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

783. The system of claim 779, wherein the execution-instruction signal includes an operation-code.

784. The system of claim 783, wherein the operation-code indicates a type of resource on which to execute.

785. The system of claim 779, wherein the execution-instruction signal includes values from a status register.

786. The system of claim 779, wherein the execution-instruction signal includes values from a priority bit status register.

787. The system of claim 779, wherein the execution-instruction signal includes a processing resource identifier.

788. The system of claim 787, wherein the processing resource identifier is an integer processing unit number.

789. The system of claim 779, wherein the execution-instruction signal includes an address.

790. The system of claim 777, wherein an other processing resource may be an delegating processing resource.

791. The system of claim 777, wherein a processing resource is an integer processing unit.

792. The system of claim 777, wherein a processing resource is a mathematical processing unit.

793. The system of claim 777, wherein a processing resource is a memory management unit.

794. The system of claim 777, wherein a processing resource is a vector processing unit.

795. The system of claim 777, wherein a processing resource is a digital signal processing unit.

796. The system of claim 777, wherein a processing resource is a graphics processing unit.

797. The system of claim 777, wherein a processing resource is an input/output controller processing unit.

798. The system of claim 777, wherein a processing resource is an execution-instruction processing cache.

799. The system of claim 777, wherein delegation occurs through an execution-instruction signal router.

800. The system of claim 799, wherein the execution-instruction signal router is a cross-point switch.

801. The system of claim 777, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

802. The system of claim 777, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

803. The system of claim 777, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

804. The system of claim 777, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

805. The system of claim 801, wherein a sleeping processing resource may be woken by a response to a request.

806. The system of claim 805, wherein the response is directed to a specific processing resource.

807. The system of claim 805, wherein the response indicates a once locked resource is unlocked.

808. The system of claim 777, wherein processing resources are communicatively disposed on a same die.

809. The system of claim 808, wherein an execution-instruction signal router is on the same die with processing resources.

810. The system of claim 777, wherein the storage-access request is obtained at a processing resource.

811. The system of claim 777, wherein information at the target memory address is designated to be local data if the target memory address is within an address range established by stack base and end register values.

812. The system of claim 777, wherein information at the target memory address is designated to be global data if the target memory address is outside an address range established by stack base and end register values.

813. The system of claim 777, wherein an apportionment is designated for each type of data for optimized access.

814. The system of claim 777, wherein the apportioned region is a hash region.

815. The system of claim 814, wherein a hash region is designated for local data.

816. The system of claim 814, wherein a hash region is designated for global data.

817. The system of claim 814, wherein the apportionment for a local data region is greater than a region for global data.

818. A system of reduced size execution of execution-instructions, comprising:

means to obtain a an execution-instruction at a processing resource;

means to append a literal constant in a literal register, if a literal flag is set by examining a special register and if there is literal prefix in the execution-instruction;

means to execute an execution-instruction with a constant in a literal register, if a literal flag is set by examining a special register and if there is no literal prefix in the execution-instruction;

means to set a literal constant flag in a status register and placing a literal constant in a literal register, if a literal flag is not set by examining a special register and if there is literal prefix in the execution-instruction;

means to execute an execution-instruction, if a literal flag is not set by examining a special register and if there is no literal prefix in the execution-instruction.

819. The system of claim 818, wherein the system is completed within a single processing cycle.

820. The system of claim 818, wherein the execution-instruction is an execution-instruction signal for execution by a type of processing resource.

821. The system of claim 820, further comprising,

means to route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

822. The system of claim 820, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

823. The system of claim 820, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

824. The system of claim 820, wherein the execution-instruction signal includes an operation-code.

825. The system of claim 824, wherein the operation-code indicates a type of resource on which to execute.

826. The system of claim 820, wherein the execution-instruction signal includes values from a status register.

827. The system of claim 820, wherein the execution-instruction signal includes values from a priority bit status register.

828. The system of claim 820, wherein the execution-instruction signal includes a processing resource identifier.

829. The system of claim 828, wherein the processing resource identifier is an integer processing unit number.

830. The system of claim 820, wherein the execution-instruction signal includes an address.

831. The system of claim 818, wherein an other processing resource may be an delegating processing resource.

832. The system of claim 818, wherein a processing resource is an integer processing unit.

833. The system of claim 818, wherein a processing resource is a mathematical processing unit.

834. The system of claim 818, wherein a processing resource is a memory management unit.

835. The system of claim 818, wherein a processing resource is a vector processing unit.

836. The system of claim 818, wherein a processing resource is a digital signal processing unit.

837. The system of claim 818, wherein a processing resource is a graphics processing unit.

838. The system of claim 818, wherein a processing resource is an input/output controller processing unit.

839. The system of claim 818, wherein a processing resource is an execution-instruction processing cache.

840. The system of claim 818, wherein delegation occurs through an execution-instruction signal router.

841. The system of claim 840, wherein the execution-instruction signal router is a cross-point switch.

842. The system of claim 818, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

843. The system of claim 818, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

844. The system of claim 818, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

845. The system of claim 818, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

846. The system of claim 842, wherein a sleeping processing resource may be woken by a response to a request.

847. The system of claim 846, wherein the response is directed to a specific processing resource.

848. The system of claim 846, wherein the response indicates a once locked resource is unlocked.

849. The system of claim 818, wherein processing resources are communicatively disposed on a same die.

850. The system of claim 849, wherein an execution-instruction signal router is on the same die with processing resources.

851. The system of claim 818, wherein the literal constant is appended by employing a shift.

852. The system of claim 818, wherein the processing resource appends the literal constant.

853. The system of claim 818, wherein the execution-instruction is executed as an extended execution of an operation-code.

854. The system of claim 853, wherein a processing resource executes the execution-instruction.

855. The system of claim 818, wherein the status register is in the processing resource.

856. The system of claim 818, wherein the literal register is in the processing resource.

857. The system of claim 818, wherein the execution-instruction is executed as a non-extended execution of an operation-code.

858. The system of claim 857, wherein a processing resource executes the execution-instruction.

859. A system of binding instructions, comprising:

means to store binding names at odd addresses;

means to generate a binding name interrupt, if a processing resource attempts to load an instruction register with an odd address value;

means to provide an operating system with a binding name interrupt;

means to perform a look-up of the odd address value that generated the binding name interrupt in a binding name table, which was established by linker;

means to replace the odd address value in an instruction register with an even address found in the binding name table.

860. The system of claim 859, wherein the interrupt is an execution-instruction signal for execution by a type of processing resource.

861. The system of claim 860, further comprising,

means to route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

862. The system of claim 860, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

863. The system of claim 860, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

864. The system of claim 860, wherein the execution-instruction signal includes an operation-code.

865. The system of claim 864, wherein the operation-code indicates a type of resource on which to execute.

866. The system of claim 860, wherein the execution-instruction signal includes values from a status register.

867. The system of claim 860, wherein the execution-instruction signal includes values from a priority bit status register.

868. The system of claim 860, wherein the execution-instruction signal includes a processing resource identifier.

869. The system of claim 868, wherein the processing resource identifier is an integer processing unit number.

870. The system of claim 860, wherein the execution-instruction signal includes an address.

871. The system of claim 859, wherein an other processing resource may be an delegating processing resource.

872. The system of claim 859, wherein a processing resource is an integer processing unit.

873. The system of claim 859, wherein a processing resource is a mathematical processing unit.

874. The system of claim 859, wherein a processing resource is a memory management unit.

875. The system of claim 859, wherein a processing resource is a vector processing unit.

876. The system of claim 859, wherein a processing resource is a digital signal processing unit.

877. The system of claim 859, wherein a processing resource is a graphics processing unit.

878. The system of claim 859, wherein a processing resource is an input/output controller processing unit.

879. The system of claim 859, wherein a processing resource is an execution-instruction processing cache.

880. The system of claim 859, wherein delegation occurs through an execution-instruction signal router.

881. The system of claim 880, wherein the execution-instruction signal router is a cross-point switch.

882. The system of claim 859, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

883. The system of claim 859, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

884. The system of claim 859, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

885. The system of claim 859, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

886. The system of claim 882, wherein a sleeping processing resource may be woken by a response to a request.

887. The system of claim 886, wherein the response is directed to a specific processing resource.

888. The system of claim 886, wherein the response indicates a once locked resource is unlocked.

889. The system of claim 859, wherein processing resources are communicatively disposed on a same die.

890. The system of claim 889, wherein an execution-instruction signal router is on the same die with processing resources.

891. The system of claim 859, wherein the binding names are stored by a linker.

892. The system of claim 859, further comprising,

means to replace the odd address value stored in an instruction stack with an instruction pointer at the even address, if late binding is being employed.

893. The system of claim 859, further comprising,

means to replace the odd address value stored in an instruction stack with an instruction pointer at the even address and replacing the odd address value in the binding name table with the even address, if dynamic binding is being employed.

894. A system of addressing registers, comprising:

means to determine if an execution-instruction operation-code provides additional register addressing information;

means to set cycle flags in a status register with the additional register addressing information;

means to examine an execution-instruction for a register address;

means to address a register specified by the register address and cycle flags.

895. The system of claim 894, wherein the system is completed within a single processing cycle.

896. The system of claim 894, wherein the execution-instruction is an execution-instruction signal for execution by a type of processing resource.

897. The system of claim 896, further comprising,

means to route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

898. The system of claim 896, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

899. The system of claim 896, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

900. The system of claim 896, wherein the execution-instruction signal includes an operation-code.

901. The system of claim 900, wherein the operation-code indicates a type of resource on which to execute.

902. The system of claim 896, wherein the execution-instruction signal includes values from a status register.

903. The system of claim 896, wherein the execution-instruction signal includes values from a priority bit status register.

904. The system of claim 896, wherein the execution-instruction signal includes a processing resource identifier.

905. The system of claim 904, wherein the processing resource identifier is an integer processing unit number.

906. The system of claim 896, wherein the execution-instruction signal includes an address.

907. The system of claim 894, wherein an other processing resource may be an delegating processing resource.

908. The system of claim 894, wherein a processing resource is an integer processing unit.

909. The system of claim 894, wherein a processing resource is a mathematical processing unit.

910. The system of claim 894, wherein a processing resource is a memory management unit.

911. The system of claim 894, wherein a processing resource is a vector processing unit.

912. The system of claim 894, wherein a processing resource is a digital signal processing unit.

913. The system of claim 894, wherein a processing resource is a graphics processing unit.

914. The system of claim 894, wherein a processing resource is an input/output controller processing unit.

915. The system of claim 894, wherein a processing resource is an execution-instruction processing cache.

916. The system of claim 894, wherein delegation occurs through an execution-instruction signal router.

917. The system of claim 916, wherein the execution-instruction signal router is a cross-point switch.

918. The system of claim 894, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

919. The system of claim 894, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

920. The system of claim 894, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

921. The system of claim 894, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

922. The system of claim 918, wherein a sleeping processing resource may be woken by a response to a request.

923. The system of claim 922, wherein the response is directed to a specific processing resource.

924. The system of claim 922, wherein the response indicates a once locked resource is unlocked.

925. The system of claim 894, wherein processing resources are communicatively disposed on a same die.

926. The system of claim 925, wherein an execution-instruction signal router is on the same die with processing resources.

927. The system of claim 894, wherein specific operation-codes address extended register sets without requiring dedicated bits set aside for register addressing in the execution-instruction.

928. The system of claim 894, wherein the register address comprises 3 bits of the execution-instruction.

929. The system of claim 894, wherein a value of a cycle flag indicates that an operation-code is a multi-cycle operation.

930. A system of sharing memory, comprising:

means to obtain a request execution-instruction from a delegate processing resource at an instruction processing cache, wherein the request includes a target memory address;

means to determine if the target address is locked based on a lock variable at the target memory address;

means to determine what type of lock is provided in the request;

means to request that a requesting processing resource sleep until it is unlocked, if the target memory address is locked.

931. The system of claim 930, wherein the system is completed within a single processing cycle.

932. The system of claim 930, wherein the request execution-instruction is an execution-instruction signal for execution by a type of processing resource.

933. The system of claim 932, further comprising,

means to route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

934. The system of claim 932, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

935. The system of claim 932, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

936. The system of claim 932, wherein the execution-instruction signal includes an operation-code.

937. The system of claim 936, wherein the operation-code indicates a type of resource on which to execute.

938. The system of claim 932, wherein the execution-instruction signal includes values from a status register.

939. The system of claim 932, wherein the execution-instruction signal includes values from a priority bit status register.

940. The system of claim 932, wherein the execution-instruction signal includes a processing resource identifier.

941. The system of claim 940, wherein the processing resource identifier is an integer processing unit number.

942. The system of claim 932, wherein the execution-instruction signal includes an address.

943. The system of claim 930, wherein an other processing resource may be an delegating processing resource.

944. The system of claim 930, wherein a processing resource is an integer processing unit.

945. The system of claim 930, wherein a processing resource is a mathematical processing unit.

946. The system of claim 930, wherein a processing resource is a memory management unit.

947. The system of claim 930, wherein a processing resource is a vector processing unit.

948. The system of claim 930, wherein a processing resource is a digital signal processing unit.

949. The system of claim 930, wherein a processing resource is a graphics processing unit.

950. The system of claim 930, wherein a processing resource is an input/output controller processing unit.

951. The system of claim 930, wherein a processing resource is an execution-instruction processing cache.

952. The system of claim 930, wherein delegation occurs through an execution-instruction signal router.

953. The system of claim 952, wherein the execution-instruction signal router is a cross-point switch.

954. The system of claim 930, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

955. The system of claim 930, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

956. The system of claim 930, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

957. The system of claim 930, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

958. The system of claim 954, wherein a sleeping processing resource may be woken by a response to a request.

959. The system of claim 958, wherein the response is directed to a specific processing resource.

960. The system of claim 958, wherein the response indicates a once locked resource is unlocked.

961. The system of claim 930, wherein processing resources are communicatively disposed on a same die.

962. The system of claim 961, wherein an execution-instruction signal router is on the same die with processing resources.

963. The system of claim 930, wherein the determination is made by the instruction processing cache's logic facilities.

964. The system of claim 930, wherein the requesting processing resource is unlocked when an unlock instruction is received that specifies the same target memory address.

965. The system of claim 930, wherein the target memory address is locked for read and write operations.

966. The system of claim 930, wherein the target memory address is locked for read only operations and the request specifies a read and write operation lock.

967. A system of sharing memory, comprising:

means to obtain a response to an execution-instruction from a delegate processing resource, wherein the response includes a target address at a processing resource;

means to determine if a target address is locked based on a lock variable at the target memory address;

means to determine value types by using a hash function;

means to update value types in a primary cache memory of a processing resource to a secondary cache memory for each value type, if each value type in a primary cache memory of a processing resource has not been updated to a secondary cache memory;

968. The system of claim 967, wherein the system is completed within a single processing cycle.

969. The system of claim 967, wherein the response is an execution-instruction signal for execution by a type of processing resource.

970. The system of claim 969, further comprising,

means to route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

971. The system of claim 969, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

972. The system of claim 969, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

973. The system of claim 969, wherein the execution-instruction signal includes an operation-code.

974. The system of claim 973, wherein the operation-code indicates a type of resource on which to execute.

975. The system of claim 969, wherein the execution-instruction signal includes values from a status register.

976. The system of claim 969, wherein the execution-instruction signal includes values from a priority bit status register.

977. The system of claim 969, wherein the execution-instruction signal includes a processing resource identifier.

978. The system of claim 977, wherein the processing resource identifier is an integer processing unit number.

979. The system of claim 969, wherein the execution-instruction signal includes an address.

980. The system of claim 967, wherein an other processing resource may be an delegating processing resource.

981. The system of claim 967, wherein a processing resource is an integer processing unit.

982. The system of claim 967, wherein a processing resource is a mathematical processing unit.

983. The system of claim 967, wherein a processing resource is a memory management unit.

984. The system of claim 967, wherein a processing resource is a vector processing unit.

985. The system of claim 967, wherein a processing resource is a digital signal processing unit.

986. The system of claim 967, wherein a processing resource is a graphics processing unit.

987. The system of claim 967, wherein a processing resource is an input/output controller processing unit.

988. The system of claim 967, wherein a processing resource is an execution-instruction processing cache.

989. The system of claim 967, wherein delegation occurs through an execution-instruction signal router.

990. The system of claim 989, wherein the execution-instruction signal router is a cross-point switch.

991. The system of claim 967, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

992. The system of claim 967, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

993. The system of claim 967, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

994. The system of claim 967, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

995. The system of claim 991, wherein a sleeping processing resource may be woken by a response to a request.

996. The system of claim 995, wherein the response is directed to a specific processing resource.

997. The system of claim 995, wherein the response indicates a once locked resource is unlocked.

998. The system of claim 967, wherein processing resources are communicatively disposed on a same die.

999. The system of claim 998, wherein an execution-instruction signal router is on the same die with processing resources.

1000. The system of claim 967, wherein the response includes a target processing resource identifier.

1001. The system of claim 967, wherein the determination is made by a processing resource.

1002. The system of claim 967, wherein the value type is a global value.

1003. The system of claim 967, wherein the secondary cache memory is a Level 2 cache memory.

1004. The system of claim 967, wherein a primary cache memory is a Level 1 cache memory.

1005. The system of claim 967, further comprising,
means to marking a cache line of the updated value type as free.

1006. The system of claim 967, further comprising,
means to update lock variables in secondary cache memory for each updated value type.

1007. The system of claim 1006, wherein the lock variable updating is performed by the secondary cache memory, which is a processing cache memory.

1008. A system of sharing memory, comprising:
means to obtain a storage request including an event variable address and execution-instruction for a processing resource to sleep;
means to determine if an event variable is set;
means to provide a reply to a processing resource bus with the event variable address and event value, which will wake an appropriate processing resource from sleep, if the event value is set not to sleep;

means to set a sleep until event value for the event variable, if the event value is set to sleep.

1009. The system of claim 1008, wherein the system is completed within a single processing cycle.

1010. The system of claim 1008, wherein the storage request is an execution-instruction signal for execution by a type of processing resource.

1011. The system of claim 1010, further comprising,

means to route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

1012. The system of claim 1010, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

1013. The system of claim 1010, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

1014. The system of claim 1010, wherein the execution-instruction signal includes an operation-code.

1015. The system of claim 1014, wherein the operation-code indicates a type of resource on which to execute.

1016. The system of claim 1010, wherein the execution-instruction signal includes values from a status register.

1017. The system of claim 1010, wherein the execution-instruction signal includes values from a priority bit status register.

1018. The system of claim 1010, wherein the execution-instruction signal includes a processing resource identifier.

1019. The system of claim 1018, wherein the processing resource identifier is an integer processing unit number.

1020. The system of claim 1010, wherein the execution-instruction signal includes an address.

1021. The system of claim 1008, wherein an other processing resource may be an delegating processing resource.

1022. The system of claim 1008, wherein a processing resource is an integer processing unit.

1023. The system of claim 1008, wherein a processing resource is a mathematical processing unit.

1024. The system of claim 1008, wherein a processing resource is a memory management unit.

1025. The system of claim 1008, wherein a processing resource is a vector processing unit.

1026. The system of claim 1008, wherein a processing resource is a digital signal processing unit.

1027. The system of claim 1008, wherein a processing resource is a graphics processing unit.

1028. The system of claim 1008, wherein a processing resource is an input/output controller processing unit.

1029. The system of claim 1008, wherein a processing resource is an execution-instruction processing cache.

1030. The system of claim 1008, wherein delegation occurs through an execution-instruction signal router.

1031. The system of claim 1030, wherein the execution-instruction signal router is a cross-point switch.

1032. The system of claim 1008, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1033. The system of claim 1008, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1034. The system of claim 1008, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1035. The system of claim 1008, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1036. The system of claim 1032, wherein a sleeping processing resource may be woken by a response to a request.

1037. The system of claim 1036, wherein the response is directed to a specific processing resource.

1038. The system of claim 1036, wherein the response indicates a once locked resource is unlocked.

1039. The system of claim 1008, wherein processing resources are communicatively disposed on a same die.

1040. The system of claim 1039, wherein an execution-instruction signal router is on the same die with processing resources.

1041. The system of claim 1008, wherein the determination is made at a processing cache.

1042. The system of claim 1008, wherein the event variable is set at the processing cache.

1043. The system of claim 1008, wherein the event variable is set in a register.

1044. The system of claim 1008, wherein the event variable is set in a target address in a processing cache.

1045. A system of sharing memory, comprising:

means to obtain a storage request including an event address at a processing cache;

means to combine a parameter with an event variable through in a processing cache;

means to provide a reply to a processing resource bus from the combination with the event address, which will wake an appropriate processing resource from sleep that is waiting on an event, if the event value is set not to sleep.

1046. The system of claim 1045, wherein the system is completed within a single processing cycle.

1047. The system of claim 1045, wherein the storage request is an execution-instruction signal for execution by a type of processing resource.

1048. The system of claim 1047, further comprising,

means to route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

1049. The system of claim 1047, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

1050. The system of claim 1047, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

1051. The system of claim 1047, wherein the execution-instruction signal includes an operation-code.

1052. The system of claim 1051, wherein the operation-code indicates a type of resource on which to execute.

1053. The system of claim 1047, wherein the execution-instruction signal includes values from a status register.

1054. The system of claim 1047, wherein the execution-instruction signal includes values from a priority bit status register.

1055. The system of claim 1047, wherein the execution-instruction signal includes a processing resource identifier.

1056. The system of claim 1055, wherein the processing resource identifier is an integer processing unit number.

1057. The system of claim 1047, wherein the execution-instruction signal includes an address.

1058. The system of claim 1045, wherein an other processing resource may be an delegating processing resource.

1059. The system of claim 1045, wherein a processing resource is an integer processing unit.

1060. The system of claim 1045, wherein a processing resource is a mathematical processing unit.

1061. The system of claim 1045, wherein a processing resource is a memory management unit.

1062. The system of claim 1045, wherein a processing resource is a vector processing unit.

1063. The system of claim 1045, wherein a processing resource is a digital signal processing unit.

1064. The system of claim 1045, wherein a processing resource is a graphics processing unit.

1065. The system of claim 1045, wherein a processing resource is an input/output controller processing unit.

1066. The system of claim 1045, wherein a processing resource is an execution-instruction processing cache.

1067. The system of claim 1045, wherein delegation occurs through an execution-instruction signal router.

1068. The system of claim 1067, wherein the execution-instruction signal router is a cross-point switch.

1069. The system of claim 1045, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1070. The system of claim 1045, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1071. The system of claim 1045, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1072. The system of claim 1045, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1073. The system of claim 1069, wherein a sleeping processing resource may be woken by a response to a request.

1074. The system of claim 1073, wherein the response is directed to a specific processing resource.

1075. The system of claim 1073, wherein the response indicates a once locked resource is unlocked.

1076. The system of claim 1045, wherein processing resources are communicatively disposed on a same die.

1077. The system of claim 1076, wherein an execution-instruction signal router is on the same die with processing resources.

1078. The system of claim 1045, wherein the parameter is a bit-mask.

1079. The system of claim 1045, wherein the combination is a logical-OR.

1080. A system of sharing memory, comprising:

means to obtain a storage request including a semaphore address at a processing cache;

means to set a semaphore variable in a processing cache at the semaphore address;

means to provide an operating system trap-call to wait on a processing queue, if the storage request includes a wait-on-semaphore instruction;

means to provide an operating system trap-call to signal on a processing queue, if the storage request includes a signal-on-semaphore instruction;

1081. The system of claim 1080, wherein the system is completed within a single processing cycle.

1082. The system of claim 1080, wherein the storage request is an execution-instruction signal for execution by a type of processing resource.

1083. The system of claim 1082, further comprising,

means to route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

1084. The system of claim 1082, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

1085. The system of claim 1082, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

1086. The system of claim 1082, wherein the execution-instruction signal includes an operation-code.

1087. The system of claim 1086, wherein the operation-code indicates a type of resource on which to execute.

1088. The system of claim 1082, wherein the execution-instruction signal includes values from a status register.

1089. The system of claim 1082, wherein the execution-instruction signal includes values from a priority bit status register.

1090. The system of claim 1082, wherein the execution-instruction signal includes a processing resource identifier.

1091. The system of claim 1090, wherein the processing resource identifier is an integer processing unit number.

1092. The system of claim 1082, wherein the execution-instruction signal includes an address.

1093. The system of claim 1080, wherein an other processing resource may be an delegating processing resource.

1094. The system of claim 1080, wherein a processing resource is an integer processing unit.

1095. The system of claim 1080, wherein a processing resource is a mathematical processing unit.

1096. The system of claim 1080, wherein a processing resource is a memory management unit.

1097. The system of claim 1080, wherein a processing resource is a vector processing unit.

1098. The system of claim 1080, wherein a processing resource is a digital signal processing unit.

1099. The system of claim 1080, wherein a processing resource is a graphics processing unit.

1100. The system of claim 1080, wherein a processing resource is an input/output controller processing unit.

1101. The system of claim 1080, wherein a processing resource is an execution-instruction processing cache.

1102. The system of claim 1080, wherein delegation occurs through an execution-instruction signal router.

1103. The system of claim 1102, wherein the execution-instruction signal router is a cross-point switch.

1104. The system of claim 1080, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1105. The system of claim 1080, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1106. The system of claim 1080, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1107. The system of claim 1080, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1108. The system of claim 1104, wherein a sleeping processing resource may be woken by a response to a request.

1109. The system of claim 1108, wherein the response is directed to a specific processing resource.

1110. The system of claim 1108, wherein the response indicates a once locked resource is unlocked.

1111. The system of claim 1080, wherein processing resources are communicatively disposed on a same die.

1112. The system of claim 1111, wherein an execution-instruction signal router is on the same die with processing resources.

1113. The system of claim 1080, wherein the trap-call causes rescheduling so that other processes can run on a particular processing resource.

1114. The system of claim 1080, wherein other processes include threads.

1115. The system of claim 1080, wherein other processes include threads.

1116. An apparatus of register addressing, comprising:
a memory, the memory for storing instructions;

a processing resource that may issue a plurality of processing instructions stored in the memory, wherein the instructions issue signals to:

obtain a register bank address;

decode an operation-code;

set a cycle flag in a status register based on the decoding of the operation-code;

access a register addressed by the cycle flag within the addressed register bank.

1117. The apparatus of claim 1116, further comprising,

employ a set number of bits in an execution-instruction to establish the number of registers within a register bank.

1118. The apparatus of claim 1116, further comprising,

employ a set number of bits in an execution-instruction to establish the number of register banks.

1119. An apparatus of setting intra-processor processing resources to sleep, comprising:

set processing resources that issue requests to delegating processing resources to sleep until a response is provided;

set processing resources waiting on shared and locked memory being accessed by other processing resources to sleep until the memory is unlocked.

1120. An apparatus of execution-instruction delegation between processing resources, comprising:

a memory, the memory for storing instructions;

a processing resource that may issue a plurality of processing instructions stored in the memory, wherein the instructions issue signals to:

share memory between processing resources,

wherein the memory itself includes instruction execution logic,

wherein the processing resources are communicatively accessible through an instruction execution router;

wherein the processing resources are on the same die;

delegate execution-instructions from originating processing resources to other processing resources.

1121. The apparatus of claim 1120, wherein the system is completed within a single processing cycle.

1122. The apparatus of claim 1120, wherein the other processing resource may be the originating processing resource.

1123. The apparatus of claim 1120, wherein a processing resource is an integer processing unit.

1124. The apparatus of claim 1120, wherein a processing resource is a mathematical processing unit.

1125. The apparatus of claim 1120, wherein a processing resource is a memory management unit.

1126. The apparatus of claim 1120, wherein a processing resource is a vector processing unit.

1127. The apparatus of claim 1120, wherein a processing resource is a digital signal processing unit.

1128. The apparatus of claim 1120, wherein a processing resource is a graphics processing unit.

1129. The apparatus of claim 1120, wherein a processing resource is an input/output controller processing unit.

1130. The apparatus of claim 1120, wherein a processing resource is an execution-instruction processing cache.

1131. The apparatus of claim 1120, wherein delegation occurs through an execution-instruction signal router.

1132. The apparatus of claim 1131, wherein the execution-instruction signal router is a cross-point switch.

1133. The apparatus of claim 1120, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1134. The apparatus of claim 1120, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1135. The apparatus of claim 1120, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1136. The apparatus of claim 1120, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1137. The apparatus of claim 1133, wherein a sleeping processing resource may be woken by a response to a request.

1138. The apparatus of claim 1137, wherein the response is directed to a specific processing resource.

1139. The apparatus of claim 1137, wherein the response indicates a once locked resource is unlocked.

1140. The apparatus of claim 1120, wherein processing resources are communicatively disposed on a same die.

1141. The apparatus of claim 1140, wherein an execution-instruction signal router is on the same die with processing resources.

1142. An apparatus of execution-instruction delegation between processing resources, comprising:

a memory, the memory for storing instructions;

a processing resource that may issue a plurality of processing instructions stored in the memory, wherein the instructions issue signals to:

obtain an execution instruction, wherein the execution instruction is obtained at a processing resource;

determine whether an operation-code within the execution instruction should be delegated to an other processing resource;

execute the execution instruction, if the operation-code within the execution instruction should not be delegated to an other processing resource;

route the execution instruction to an other processing resource, if the operation-code within the execution instruction is for an other processing resource.

1143. The apparatus of claim 1142, wherein the system is completed within a single processing cycle.

1144. The apparatus of claim 1142, wherein the routed execution instruction is a request for execution by a type of processing resource.

1145. The apparatus of claim 1144, wherein the request includes an operation-code.

1146. The apparatus of claim 1145, wherein the operation-code indicates a type of resource on which to execute.

1147. The apparatus of claim 1144, wherein the request includes values from a status register.

1148. The apparatus of claim 1144, wherein the request includes values from a priority bit status register.

1149. The apparatus of claim 1144, wherein the request includes a processing resource identifier.

1150. The apparatus of claim 1149, wherein the processing resource identifier is an integer processing unit number.

1151. The apparatus of claim 1144, wherein the request is obtained by an execution-instruction signal router.

1152. The apparatus of claim 1144, wherein the request includes an address.

1153. The apparatus of claim 1142, wherein the operation-code indicates a type of resource on which to execute.

1154. The apparatus of claim 1142, wherein the other processing resource may be the originating processing resource.

1155. The apparatus of claim 1142, wherein a processing resource is an integer processing unit.

1156. The apparatus of claim 1142, wherein a processing resource is a mathematical processing unit.

1157. The apparatus of claim 1142, wherein a processing resource is a memory management unit.

1158. The apparatus of claim 1142, wherein a processing resource is a vector processing unit.

1159. The apparatus of claim 1142, wherein a processing resource is a digital signal processing unit.

1160. The apparatus of claim 1142, wherein a processing resource is a graphics processing unit.

1161. The apparatus of claim 1142, wherein a processing resource is an input/output controller processing unit.

1162. The apparatus of claim 1142, wherein a processing resource is an execution-instruction processing cache.

1163. The apparatus of claim 1142, wherein routing occurs through an execution-instruction signal router.

1164. The apparatus of claim 1163, wherein the execution-instruction signal router is a cross-point switch.

1165. The apparatus of claim 1142, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1166. The apparatus of claim 1142, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1167. The apparatus of claim 1142, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1168. The apparatus of claim 1142, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1169. The apparatus of claim 1142, wherein processing resources are communicatively disposed on a same die.

1170. The apparatus of claim 1169, wherein an execution-instruction signal router is on the same die with processing resources.

1171. An apparatus of execution-instruction delegation between processing resources, comprising:

a memory, the memory for storing instructions;

a processing resource that may issue a plurality of processing instructions stored in the memory, wherein the instructions issue signals to:

obtain an request signal, wherein the request signal is from a requesting processing resource;

determine a priority dead-lock-avoidance value, wherein the priority dead-lock value is used to select among multiple requests with equal priority;

examine a request priority value for each submitted request;

select a request with a highest priority value, if more than one processing resource requests a same target resource;

provide a selected request to a target processing resource.

1172. The apparatus of claim 1171, wherein the system is completed within a single processing cycle.

1173. The apparatus of claim 1171, wherein the request signal is a request for execution by a type of processing resource.

1174. The apparatus of claim 1173, further comprising,

route the request to an other processing resource, if an operation-code within the request is for an other processing resource.

1175. The apparatus of claim 1173, wherein the request is obtained at an execution-instruction signal router.

1176. The apparatus of claim 1173, wherein the execution-instruction signal is provided by a delegating processing resource via an execution-instruction signal router.

1177. The apparatus of claim 1173, wherein the request includes an operation-code.

1178. The apparatus of claim 1177, wherein the operation-code indicates a type of resource on which to execute.

1179. The apparatus of claim 1173, wherein the request includes values from a status register.

1180. The apparatus of claim 1173, wherein the request includes values from a priority bit status register.

1181. The apparatus of claim 1173, wherein the request includes a processing resource identifier.

1182. The apparatus of claim 1181, wherein the processing resource identifier is an integer processing unit number.

1183. The apparatus of claim 1173, wherein the request includes an address.

1184. The apparatus of claim 1171, wherein the other processing resource may be the originating processing resource.

1185. The apparatus of claim 1171, wherein a processing resource is an integer processing unit.

1186. The apparatus of claim 1171, wherein a processing resource is a mathematical processing unit.

1187. The apparatus of claim 1171, wherein a processing resource is a memory management unit.

1188. The apparatus of claim 1171, wherein a processing resource is a vector processing unit.

1189. The apparatus of claim 1171, wherein a processing resource is a digital signal processing unit.

1190. The apparatus of claim 1171, wherein a processing resource is a graphics processing unit.

1191. The apparatus of claim 1171, wherein a processing resource is an input/output controller processing unit.

1192. The apparatus of claim 1171, wherein a processing resource is an execution-instruction processing cache.

1193. The apparatus of claim 1171, wherein delegation occurs through an execution-instruction signal router.

1194. The apparatus of claim 1193, wherein the execution-instruction signal router is a cross-point switch.

1195. The apparatus of claim 1171, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1196. The apparatus of claim 1171, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1197. The apparatus of claim 1171, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1198. The apparatus of claim 1171, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1199. The apparatus of claim 1195, wherein a sleeping processing resource may be woken by a response to a request.

1200. The apparatus of claim 1199, wherein the response is directed to a specific processing resource.

1201. The apparatus of claim 1199, wherein the response indicates a once locked resource is unlocked.

1202. The apparatus of claim 1171, wherein processing resources are communicatively disposed on a same die.

1203. The apparatus of claim 1202, wherein an execution-instruction signal router is on the same die with processing resources.

1204. The apparatus of claim 1171, wherein the priority dead-lock-avoidance value is iterated.

1205. The apparatus of claim 1171, further comprising,

select a request, if not more than one processing resource requests a same target resource.

1206. The apparatus of claim 1171, further comprising,

select the request with a highest priority value, if not more than one request has a same highest priority value;

select a request based on the priority dead-lock-avoidance value, if more than one request has a same highest priority value.

1207. The apparatus of claim 1171, further comprising,

provide a request granted acknowledgement to the requesting processing resource whose request was selected.

1208. The apparatus of claim 1171, further comprising,

clear the operation-code in the requesting processing resource whose request was selected.

1209. An apparatus of execution-instruction delegation between processing resources, comprising:

a memory, the memory for storing instructions;

a processing resource that may issue a plurality of processing instructions stored in the memory, wherein the instructions issue signals to:

process a request execution-instruction, wherein the request execution-instruction includes a requesting processing resource identifier;

prepare a response into a result register, wherein the response includes the requesting processing resource identifier;

present the response to all processing resources.

1210. The apparatus of claim 1209, wherein the system is completed within a single processing cycle.

1211. The apparatus of claim 1209, wherein the request execution-instruction is a request for execution by a type of processing resource.

1212. The apparatus of claim 1211, further comprising,

route the request to an other processing resource, if an operation-code within the request is for an other processing resource.

1213. The apparatus of claim 1211, wherein the request is obtained at an execution-instruction signal router.

1214. The apparatus of claim 1211, wherein the execution-instruction signal is provided by a delegating processing resource via an execution-instruction signal router.

1215. The apparatus of claim 1211, wherein the request includes an operation-code.

1216. The apparatus of claim 1215, wherein the operation-code indicates a type of resource on which to execute.

1217. The apparatus of claim 1211, wherein the request includes values from a status register.

1218. The apparatus of claim 1211, wherein the request includes values from a priority bit status register.

1219. The apparatus of claim 1211, wherein the request includes a processing resource identifier.

1220. The apparatus of claim 1219, wherein the processing resource identifier is an integer processing unit number.

1221. The apparatus of claim 1211, wherein the request includes an address.

1222. The apparatus of claim 1209, wherein the other processing resource may be the originating processing resource.

1223. The apparatus of claim 1209, wherein a processing resource is an integer processing unit.

1224. The apparatus of claim 1209, wherein a processing resource is a mathematical processing unit.

1225. The apparatus of claim 1209, wherein a processing resource is a memory management unit.

1226. The apparatus of claim 1209, wherein a processing resource is a vector processing unit.

1227. The apparatus of claim 1209, wherein a processing resource is a digital signal processing unit.

1228. The apparatus of claim 1209, wherein a processing resource is a graphics processing unit.

1229. The apparatus of claim 1209, wherein a processing resource is an input/output controller processing unit.

1230. The apparatus of claim 1209, wherein a processing resource is an execution-instruction processing cache.

1231. The apparatus of claim 1209, wherein delegation occurs through an execution-instruction signal router.

1232. The apparatus of claim 1231, wherein the execution-instruction signal router is a cross-point switch.

1233. The apparatus of claim 1209, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1234. The apparatus of claim 1209, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1235. The apparatus of claim 1209, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1236. The apparatus of claim 1209, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1237. The apparatus of claim 1233, wherein a sleeping processing resource may be woken by a response to a request.

1238. The apparatus of claim 1237, wherein the response is directed to a specific processing resource.

1239. The apparatus of claim 1237, wherein the response indicates a once locked resource is unlocked.

1240. The apparatus of claim 1209, wherein processing resources are communicatively disposed on a same die.

1241. The apparatus of claim 1240, wherein an execution-instruction signal router is on the same die with processing resources.

1242. An apparatus of execution-instruction delegation between processing resources, comprising:

a memory, the memory for storing instructions;

a processing resource that may issue a plurality of processing instructions stored in the memory, wherein the instructions issue signals to:

obtain a result execution-instruction from a delegate processing resource, wherein the result includes a requesting processing resource identifier;

wake the requesting processing resource, if the requesting processing resource identifier identifies the instant processing resource;

wake a processing resource, if a processing resource is waiting to be unlocked and if the obtained result includes an unlock flag and if an address in the result matches a locked address of an instant processing resource.

1243. The apparatus of claim 1242, wherein the system is completed within a single processing cycle.

1244. The apparatus of claim 1242, wherein the result execution-instruction is an execution-instruction signal for execution by a type of processing resource.

1245. The apparatus of claim 1244, further comprising,

route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

1246. The apparatus of claim 1244, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

1247. The apparatus of claim 1244, wherein the execution-instruction signal is provided by a delegating processing resource via an execution-instruction signal router.

1248. The apparatus of claim 1244, wherein the execution-instruction signal includes an operation-code.

1249. The apparatus of claim 1248, wherein the operation-code indicates a type of resource on which to execute.

1250. The apparatus of claim 1244, wherein the execution-instruction signal includes values from a status register.

1251. The apparatus of claim 1244, wherein the execution-instruction signal includes values from a priority bit status register.

1252. The apparatus of claim 1244, wherein the execution-instruction signal includes a processing resource identifier.

1253. The apparatus of claim 1252, wherein the processing resource identifier is an integer processing unit number.

1254. The apparatus of claim 1244, wherein the execution-instruction signal includes an address.

1255. The apparatus of claim 1242, wherein an other processing resource may be an delegating processing resource.

1256. The apparatus of claim 1242, wherein a processing resource is an integer processing unit.

1257. The apparatus of claim 1242, wherein a processing resource is a mathematical processing unit.

1258. The apparatus of claim 1242, wherein a processing resource is a memory management unit.

1259. The apparatus of claim 1242, wherein a processing resource is a vector processing unit.

1260. The apparatus of claim 1242, wherein a processing resource is a digital signal processing unit.

1261. The apparatus of claim 1242, wherein a processing resource is a graphics processing unit.

1262. The apparatus of claim 1242, wherein a processing resource is an input/output controller processing unit.

1263. The apparatus of claim 1242, wherein a processing resource is an execution-instruction processing cache.

1264. The apparatus of claim 1242, wherein delegation occurs through an execution-instruction signal router.

1265. The apparatus of claim 1264, wherein the execution-instruction signal router is a cross-point switch.

1266. The apparatus of claim 1242, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1267. The apparatus of claim 1242, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1268. The apparatus of claim 1242, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1269. The apparatus of claim 1242, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1270. The apparatus of claim 1266, wherein a sleeping processing resource may be woken by a response to a request.

1271. The apparatus of claim 1270, wherein the response is directed to a specific processing resource.

1272. The apparatus of claim 1270, wherein the response indicates a once locked resource is unlocked.

1273. The apparatus of claim 1242, wherein processing resources are communicatively disposed on a same die.

1274. The apparatus of claim 1273, wherein an execution-instruction signal router is on the same die with processing resources.

1275. The apparatus of claim 1242, further comprising,

execute a next execution-instruction, if the requesting processing resource identifier identifies the instant processing resource;

execute an execution-instruction that failed to execute, if a processing resource is waiting to be unlocked and if the obtained result includes an unlock flag and if an address in the result matches a locked address of an instant processing resource.

1276. An apparatus of memory access optimization, comprising:

a memory, the memory for storing instructions;

a processing resource that may issue a plurality of processing instructions stored in the memory, wherein the instructions issue signals to:

obtain a storage-access request from a delegated processing resource, wherein the request includes a target memory address;

determine the target memory address from the request;

compare the target memory address with register values, wherein register values are used to establish a data type of the target memory address for subsequent storage in an apportioned region of cache memory;

obtain information from the target memory address;

store the information in an apportioned region of cache memory.

1277. The apparatus of claim 1276, wherein the system is completed within a single processing cycle.

1278. The apparatus of claim 1276, wherein the storage-access request is an execution-instruction signal for execution by a type of processing resource.

1279. The apparatus of claim 1278, further comprising,

route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

1280. The apparatus of claim 1278, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

1281. The apparatus of claim 1278, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

1282. The apparatus of claim 1278, wherein the execution-instruction signal includes an operation-code.

1283. The apparatus of claim 1282, wherein the operation-code indicates a type of resource on which to execute.

1284. The apparatus of claim 1278, wherein the execution-instruction signal includes values from a status register.

1285. The apparatus of claim 1278, wherein the execution-instruction signal includes values from a priority bit status register.

1286. The apparatus of claim 1278, wherein the execution-instruction signal includes a processing resource identifier.

1287. The apparatus of claim 1286, wherein the processing resource identifier is an integer processing unit number.

1288. The apparatus of claim 1278, wherein the execution-instruction signal includes an address.

1289. The apparatus of claim 1276, wherein an other processing resource may be an delegating processing resource.

1290. The apparatus of claim 1276, wherein a processing resource is an integer processing unit.

1291. The apparatus of claim 1276, wherein a processing resource is a mathematical processing unit.

1292. The apparatus of claim 1276, wherein a processing resource is a memory management unit.

1293. The apparatus of claim 1276, wherein a processing resource is a vector processing unit.

1294. The apparatus of claim 1276, wherein a processing resource is a digital signal processing unit.

1295. The apparatus of claim 1276, wherein a processing resource is a graphics processing unit.

1296. The apparatus of claim 1276, wherein a processing resource is an input/output controller processing unit.

1297. The apparatus of claim 1276, wherein a processing resource is an execution-instruction processing cache.

1298. The apparatus of claim 1276, wherein delegation occurs through an execution-instruction signal router.

1299. The apparatus of claim 1298, wherein the execution-instruction signal router is a cross-point switch.

1300. The apparatus of claim 1276, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1301. The apparatus of claim 1276, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1302. The apparatus of claim 1276, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1303. The apparatus of claim 1276, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1304. The apparatus of claim 1300, wherein a sleeping processing resource may be woken by a response to a request.

1305. The apparatus of claim 1304, wherein the response is directed to a specific processing resource.

1306. The apparatus of claim 1304, wherein the response indicates a once locked resource is unlocked.

1307. The apparatus of claim 1276, wherein processing resources are communicatively disposed on a same die.

1308. The apparatus of claim 1307, wherein an execution-instruction signal router is on the same die with processing resources.

1309. The apparatus of claim 1276, wherein the storage-access request is obtained at a processing resource.

1310. The apparatus of claim 1276, wherein information at the target memory address is designated to be local data if the target memory address is within an address range established by stack base and end register values.

1311. The apparatus of claim 1276, wherein information at the target memory address is designated to be global data if the target memory address is outside an address range established by stack base and end register values.

1312. The apparatus of claim 1276, wherein an apportionment is designated for each type of data for optimized access.

1313. The apparatus of claim 1276, wherein the apportioned region is a hash region.

1314. The apparatus of claim 1313, wherein a hash region is designated for local data.

1315. The apparatus of claim 1313, wherein a hash region is designated for global data.

1316. The apparatus of claim 1313, wherein the apportionment for a local data region is greater than a region for global data.

1317. An apparatus of reduced size execution of execution-instructions, comprising:

a memory, the memory for storing instructions;

a processing resource that may issue a plurality of processing instructions stored in the memory, wherein the instructions issue signals to:

obtain a an execution-instruction at a processing resource;

append a literal constant in a literal register, if a literal flag is set by examining a special register and if there is literal prefix in the execution-instruction;

execute an execution-instruction with a constant in a literal register, if a literal flag is set by examining a special register and if there is no literal prefix in the execution-instruction;

set a literal constant flag in a status register and placing a literal constant in a literal register, if a literal flag is not set by examining a special register and if there is literal prefix in the execution-instruction;

execute an execution-instruction, if a literal flag is not set by examining a special register and if there is no literal prefix in the execution-instruction.

1318. The apparatus of claim 1317, wherein the system is completed within a single processing cycle.

1319. The apparatus of claim 1317, wherein the execution-instruction is an execution-instruction signal for execution by a type of processing resource.

1320. The apparatus of claim 1319, further comprising,

route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

1321. The apparatus of claim 1319, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

1322. The apparatus of claim 1319, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

1323. The apparatus of claim 1319, wherein the execution-instruction signal includes an operation-code.

1324. The apparatus of claim 1323, wherein the operation-code indicates a type of resource on which to execute.

1325. The apparatus of claim 1319, wherein the execution-instruction signal includes values from a status register.

1326. The apparatus of claim 1319, wherein the execution-instruction signal includes values from a priority bit status register.

1327. The apparatus of claim 1319, wherein the execution-instruction signal includes a processing resource identifier.

1328. The apparatus of claim 1327, wherein the processing resource identifier is an integer processing unit number.

1329. The apparatus of claim 1319, wherein the execution-instruction signal includes an address.

1330. The apparatus of claim 1317, wherein an other processing resource may be an delegating processing resource.

1331. The apparatus of claim 1317, wherein a processing resource is an integer processing unit.

1332. The apparatus of claim 1317, wherein a processing resource is a mathematical processing unit.

1333. The apparatus of claim 1317, wherein a processing resource is a memory management unit.

1334. The apparatus of claim 1317, wherein a processing resource is a vector processing unit.

1335. The apparatus of claim 1317, wherein a processing resource is a digital signal processing unit.

1336. The apparatus of claim 1317, wherein a processing resource is a graphics processing unit.

1337. The apparatus of claim 1317, wherein a processing resource is an input/output controller processing unit.

1338. The apparatus of claim 1317, wherein a processing resource is an execution-instruction processing cache.

1339. The apparatus of claim 1317, wherein delegation occurs through an execution-instruction signal router.

1340. The apparatus of claim 1339, wherein the execution-instruction signal router is a cross-point switch.

1341. The apparatus of claim 1317, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1342. The apparatus of claim 1317, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1343. The apparatus of claim 1317, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1344. The apparatus of claim 1317, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1345. The apparatus of claim 1341, wherein a sleeping processing resource may be woken by a response to a request.

1346. The apparatus of claim 1345, wherein the response is directed to a specific processing resource.

1347. The apparatus of claim 1345, wherein the response indicates a once locked resource is unlocked.

1348. The apparatus of claim 1317, wherein processing resources are communicatively disposed on a same die.

1349. The apparatus of claim 1348, wherein an execution-instruction signal router is on the same die with processing resources.

1350. The apparatus of claim 1317, wherein the literal constant is appended by employing a shift.

1351. The apparatus of claim 1317, wherein the processing resource appends the literal constant.

1352. The apparatus of claim 1317, wherein the execution-instruction is executed as an extended execution of an operation-code.

1353. The apparatus of claim 1352, wherein a processing resource executes the execution-instruction.

1354. The apparatus of claim 1317, wherein the status register is in the processing resource.

1355. The apparatus of claim 1317, wherein the literal register is in the processing resource.

1356. The apparatus of claim 1317, wherein the execution-instruction is executed as a non-extended execution of an operation-code.

1357. The apparatus of claim 1356, wherein a processing resource executes the execution-instruction.

1358. An apparatus of binding instructions, comprising:

a memory, the memory for storing instructions;

a processing resource that may issue a plurality of processing instructions stored in the memory, wherein the instructions issue signals to:

store binding names at odd addresses;

generate a binding name interrupt, if a processing resource attempts to load an instruction register with an odd address value;

provide an operating system with a binding name interrupt;

perform a look-up of the odd address value that generated the binding name interrupt in a binding name table, which was established by linker;

replace the odd address value in an instruction register with an even address found in the binding name table.

1359. The apparatus of claim 1358, wherein the interrupt is an execution-instruction signal for execution by a type of processing resource.

1360. The apparatus of claim 1359, further comprising,

route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

1361. The apparatus of claim 1359, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

1362. The apparatus of claim 1359, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

1363. The apparatus of claim 1359, wherein the execution-instruction signal includes an operation-code.

1364. The apparatus of claim 1363, wherein the operation-code indicates a type of resource on which to execute.

1365. The apparatus of claim 1359, wherein the execution-instruction signal includes values from a status register.

1366. The apparatus of claim 1359, wherein the execution-instruction signal includes values from a priority bit status register.

1367. The apparatus of claim 1359, wherein the execution-instruction signal includes a processing resource identifier.

1368. The apparatus of claim 1367, wherein the processing resource identifier is an integer processing unit number.

1369. The apparatus of claim 1359, wherein the execution-instruction signal includes an address.

1370. The apparatus of claim 1358, wherein an other processing resource may be an delegating processing resource.

1371. The apparatus of claim 1358, wherein a processing resource is an integer processing unit.

1372. The apparatus of claim 1358, wherein a processing resource is a mathematical processing unit.

1373. The apparatus of claim 1358, wherein a processing resource is a memory management unit.

1374. The apparatus of claim 1358, wherein a processing resource is a vector processing unit.

1375. The apparatus of claim 1358, wherein a processing resource is a digital signal processing unit.

1376. The apparatus of claim 1358, wherein a processing resource is a graphics processing unit.

1377. The apparatus of claim 1358, wherein a processing resource is an input/output controller processing unit.

1378. The apparatus of claim 1358, wherein a processing resource is an execution-instruction processing cache.

1379. The apparatus of claim 1358, wherein delegation occurs through an execution-instruction signal router.

1380. The apparatus of claim 1379, wherein the execution-instruction signal router is a cross-point switch.

1381. The apparatus of claim 1358, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1382. The apparatus of claim 1358, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1383. The apparatus of claim 1358, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1384. The apparatus of claim 1358, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1385. The apparatus of claim 1381, wherein a sleeping processing resource may be woken by a response to a request.

1386. The apparatus of claim 1385, wherein the response is directed to a specific processing resource.

1387. The apparatus of claim 1385, wherein the response indicates a once locked resource is unlocked.

1388. The apparatus of claim 1358, wherein processing resources are communicatively disposed on a same die.

1389. The apparatus of claim 1388, wherein an execution-instruction signal router is on the same die with processing resources.

1390. The apparatus of claim 1358, wherein the binding names are stored by a linker.

1391. The apparatus of claim 1358, further comprising,

replace the odd address value stored in an instruction stack with an instruction pointer at the even address, if late binding is being employed.

1392. The apparatus of claim 1358, further comprising,

replace the odd address value stored in an instruction stack with an instruction pointer at the even address and replacing the odd address value in the binding name table with the even address, if dynamic binding is being employed.

1393. An apparatus of addressing registers, comprising:

a memory, the memory for storing instructions;

a processing resource that may issue a plurality of processing instructions stored in the memory, wherein the instructions issue signals to:

determine if an execution-instruction operation-code provides additional register addressing information;

set cycle flags in a status register with the additional register addressing information;

examine an execution-instruction for a register address;

address a register specified by the register address and cycle flags.

1394. The apparatus of claim 1393, wherein the system is completed within a single processing cycle.

1395. The apparatus of claim 1393, wherein the execution-instruction is an execution-instruction signal for execution by a type of processing resource.

1396. The apparatus of claim 1395, further comprising,

route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

1397. The apparatus of claim 1395, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

1398. The apparatus of claim 1395, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

1399. The apparatus of claim 1395, wherein the execution-instruction signal includes an operation-code.

1400. The apparatus of claim 1399, wherein the operation-code indicates a type of resource on which to execute.

1401. The apparatus of claim 1395, wherein the execution-instruction signal includes values from a status register.

1402. The apparatus of claim 1395, wherein the execution-instruction signal includes values from a priority bit status register.

1403. The apparatus of claim 1395, wherein the execution-instruction signal includes a processing resource identifier.

1404. The apparatus of claim 1403, wherein the processing resource identifier is an integer processing unit number.

1405. The apparatus of claim 1395, wherein the execution-instruction signal includes an address.

1406. The apparatus of claim 1393, wherein an other processing resource may be an delegating processing resource.

1407. The apparatus of claim 1393, wherein a processing resource is an integer processing unit.

1408. The apparatus of claim 1393, wherein a processing resource is a mathematical processing unit.

1409. The apparatus of claim 1393, wherein a processing resource is a memory management unit.

1410. The apparatus of claim 1393, wherein a processing resource is a vector processing unit.

1411. The apparatus of claim 1393, wherein a processing resource is a digital signal processing unit.

1412. The apparatus of claim 1393, wherein a processing resource is a graphics processing unit.

1413. The apparatus of claim 1393, wherein a processing resource is an input/output controller processing unit.

1414. The apparatus of claim 1393, wherein a processing resource is an execution-instruction processing cache.

1415. The apparatus of claim 1393, wherein delegation occurs through an execution-instruction signal router.

1416. The apparatus of claim 1415, wherein the execution-instruction signal router is a cross-point switch.

1417. The apparatus of claim 1393, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1418. The apparatus of claim 1393, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1419. The apparatus of claim 1393, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1420. The apparatus of claim 1393, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1421. The apparatus of claim 1417, wherein a sleeping processing resource may be woken by a response to a request.

1422. The apparatus of claim 1421, wherein the response is directed to a specific processing resource.

1423. The apparatus of claim 1421, wherein the response indicates a once locked resource is unlocked.

1424. The apparatus of claim 1393, wherein processing resources are communicatively disposed on a same die.

1425. The apparatus of claim 1424, wherein an execution-instruction signal router is on the same die with processing resources.

1426. The apparatus of claim 1393, wherein specific operation-codes address extended register sets without requiring dedicated bits set aside for register addressing in the execution-instruction.

1427. The apparatus of claim 1393, wherein the register address comprises 3 bits of the execution-instruction.

1428. The apparatus of claim 1393, wherein a value of a cycle flag indicates that an operation-code is a multi-cycle operation.

1429. An apparatus of sharing memory, comprising:

a memory, the memory for storing instructions;

a processing resource that may issue a plurality of processing instructions stored in the memory, wherein the instructions issue signals to:

obtain a request execution-instruction from a delegate processing resource at an instruction processing cache, wherein the request includes a target memory address;

determine if the target address is locked based on a lock variable at the target memory address;

determine what type of lock is provided in the request;

request that a requesting processing resource sleep until it is unlocked, if the target memory address is locked.

1430. The apparatus of claim 1429, wherein the system is completed within a single processing cycle.

1431. The apparatus of claim 1429, wherein the request execution-instruction is an execution-instruction signal for execution by a type of processing resource.

1432. The apparatus of claim 1431, further comprising,

route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

1433. The apparatus of claim 1431, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

1434. The apparatus of claim 1431, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

1435. The apparatus of claim 1431, wherein the execution-instruction signal includes an operation-code.

1436. The apparatus of claim 1435, wherein the operation-code indicates a type of resource on which to execute.

1437. The apparatus of claim 1431, wherein the execution-instruction signal includes values from a status register.

1438. The apparatus of claim 1431, wherein the execution-instruction signal includes values from a priority bit status register.

1439. The apparatus of claim 1431, wherein the execution-instruction signal includes a processing resource identifier.

1440. The apparatus of claim 1439, wherein the processing resource identifier is an integer processing unit number.

1441. The apparatus of claim 1431, wherein the execution-instruction signal includes an address.

1442. The apparatus of claim 1429, wherein an other processing resource may be an delegating processing resource.

1443. The apparatus of claim 1429, wherein a processing resource is an integer processing unit.

1444. The apparatus of claim 1429, wherein a processing resource is a mathematical processing unit.

1445. The apparatus of claim 1429, wherein a processing resource is a memory management unit.

1446. The apparatus of claim 1429, wherein a processing resource is a vector processing unit.

1447. The apparatus of claim 1429, wherein a processing resource is a digital signal processing unit.

1448. The apparatus of claim 1429, wherein a processing resource is a graphics processing unit.

1449. The apparatus of claim 1429, wherein a processing resource is an input/output controller processing unit.

1450. The apparatus of claim 1429, wherein a processing resource is an execution-instruction processing cache.

1451. The apparatus of claim 1429, wherein delegation occurs through an execution-instruction signal router.

1452. The apparatus of claim 1451, wherein the execution-instruction signal router is a cross-point switch.

1453. The apparatus of claim 1429, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1454. The apparatus of claim 1429, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1455. The apparatus of claim 1429, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1456. The apparatus of claim 1429, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1457. The apparatus of claim 1453, wherein a sleeping processing resource may be woken by a response to a request.

1458. The apparatus of claim 1457, wherein the response is directed to a specific processing resource.

1459. The apparatus of claim 1457, wherein the response indicates a once locked resource is unlocked.

1460. The apparatus of claim 1429, wherein processing resources are communicatively disposed on a same die.

1461. The apparatus of claim 1460, wherein an execution-instruction signal router is on the same die with processing resources.

1462. The apparatus of claim 1429, wherein the determination is made by the instruction processing cache's logic facilities.

1463. The apparatus of claim 1429, wherein the requesting processing resource is unlocked when an unlock instruction is received that specifies the same target memory address.

1464. The apparatus of claim 1429, wherein the target memory address is locked for read and write operations.

1465. The apparatus of claim 1429, wherein the target memory address is locked for read only operations and the request specifies a read and write operation lock.

1466. An apparatus of sharing memory, comprising:

a memory, the memory for storing instructions;

a processing resource that may issue a plurality of processing instructions stored in the memory, wherein the instructions issue signals to:

obtain a response to an execution-instruction from a delegate processing resource, wherein the response includes a target address at a processing resource;

determine if a target address is locked based on a lock variable at the target memory address;

determine value types by using a hash function;

update value types in a primary cache memory of a processing resource to a secondary cache memory for each value type, if each value type in a primary cache memory of a processing resource has not been updated to a secondary cache memory;

1467. The apparatus of claim 1466, wherein the system is completed within a single processing cycle.

1468. The apparatus of claim 1466, wherein the response is an execution-instruction signal for execution by a type of processing resource.

1469. The apparatus of claim 1468, further comprising,

route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

1470. The apparatus of claim 1468, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

1471. The apparatus of claim 1468, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

1472. The apparatus of claim 1468, wherein the execution-instruction signal includes an operation-code.

1473. The apparatus of claim 1472, wherein the operation-code indicates a type of resource on which to execute.

1474. The apparatus of claim 1468, wherein the execution-instruction signal includes values from a status register.

1475. The apparatus of claim 1468, wherein the execution-instruction signal includes values from a priority bit status register.

1476. The apparatus of claim 1468, wherein the execution-instruction signal includes a processing resource identifier.

1477. The apparatus of claim 1476, wherein the processing resource identifier is an integer processing unit number.

1478. The apparatus of claim 1468, wherein the execution-instruction signal includes an address.

1479. The apparatus of claim 1466, wherein an other processing resource may be an delegating processing resource.

1480. The apparatus of claim 1466, wherein a processing resource is an integer processing unit.

1481. The apparatus of claim 1466, wherein a processing resource is a mathematical processing unit.

1482. The apparatus of claim 1466, wherein a processing resource is a memory management unit.

1483. The apparatus of claim 1466, wherein a processing resource is a vector processing unit.

1484. The apparatus of claim 1466, wherein a processing resource is a digital signal processing unit.

1485. The apparatus of claim 1466, wherein a processing resource is a graphics processing unit.

1486. The apparatus of claim 1466, wherein a processing resource is an input/output controller processing unit.

1487. The apparatus of claim 1466, wherein a processing resource is an execution-instruction processing cache.

1488. The apparatus of claim 1466, wherein delegation occurs through an execution-instruction signal router.

1489. The apparatus of claim 1488, wherein the execution-instruction signal router is a cross-point switch.

1490. The apparatus of claim 1466, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1491. The apparatus of claim 1466, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1492. The apparatus of claim 1466, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1493. The apparatus of claim 1466, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1494. The apparatus of claim 1490, wherein a sleeping processing resource may be woken by a response to a request.

1495. The apparatus of claim 1494, wherein the response is directed to a specific processing resource.

1496. The apparatus of claim 1494, wherein the response indicates a once locked resource is unlocked.

1497. The apparatus of claim 1466, wherein processing resources are communicatively disposed on a same die.

1498. The apparatus of claim 1497, wherein an execution-instruction signal router is on the same die with processing resources.

1499. The apparatus of claim 1466, wherein the response includes a target processing resource identifier.

1500. The apparatus of claim 1466, wherein the determination is made by a processing resource.

1501. The apparatus of claim 1466, wherein the value type is a global value.

1502. The apparatus of claim 1466, wherein the secondary cache memory is a Level 2 cache memory.

1503. The apparatus of claim 1466, wherein a primary cache memory is a Level 1 cache memory.

1504. The apparatus of claim 1466, further comprising,
marking a cache line of the updated value type as free.

1505. The apparatus of claim 1466, further comprising,
update lock variables in secondary cache memory for each updated value type.

1506. The apparatus of claim 1505, wherein the lock variable updating is performed by the secondary cache memory, which is a processing cache memory.

1507. An apparatus of sharing memory, comprising:

a memory, the memory for storing instructions;

a processing resource that may issue a plurality of processing instructions stored in the memory, wherein the instructions issue signals to:

obtain a storage request including an event variable address and execution-instruction for a processing resource to sleep;

determine if an event variable is set;

provide a reply to a processing resource bus with the event variable address and event value, which will wake an appropriate processing resource from sleep, if the event value is set not to sleep;

set a sleep until event value for the event variable, if the event value is set to sleep.

1508. The apparatus of claim 1507, wherein the system is completed within a single processing cycle.

1509. The apparatus of claim 1507, wherein the storage request is an execution-instruction signal for execution by a type of processing resource.

1510. The apparatus of claim 1509, further comprising,

route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

1511. The apparatus of claim 1509, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

1512. The apparatus of claim 1509, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

1513. The apparatus of claim 1509, wherein the execution-instruction signal includes an operation-code.

1514. The apparatus of claim 1513, wherein the operation-code indicates a type of resource on which to execute.

1515. The apparatus of claim 1509, wherein the execution-instruction signal includes values from a status register.

1516. The apparatus of claim 1509, wherein the execution-instruction signal includes values from a priority bit status register.

1517. The apparatus of claim 1509, wherein the execution-instruction signal includes a processing resource identifier.

1518. The apparatus of claim 1517, wherein the processing resource identifier is an integer processing unit number.

1519. The apparatus of claim 1509, wherein the execution-instruction signal includes an address.

1520. The apparatus of claim 1507, wherein an other processing resource may be an delegating processing resource.

1521. The apparatus of claim 1507, wherein a processing resource is an integer processing unit.

1522. The apparatus of claim 1507, wherein a processing resource is a mathematical processing unit.

1523. The apparatus of claim 1507, wherein a processing resource is a memory management unit.

1524. The apparatus of claim 1507, wherein a processing resource is a vector processing unit.

1525. The apparatus of claim 1507, wherein a processing resource is a digital signal processing unit.

1526. The apparatus of claim 1507, wherein a processing resource is a graphics processing unit.

1527. The apparatus of claim 1507, wherein a processing resource is an input/output controller processing unit.

1528. The apparatus of claim 1507, wherein a processing resource is an execution-instruction processing cache.

1529. The apparatus of claim 1507, wherein delegation occurs through an execution-instruction signal router.

1530. The apparatus of claim 1529, wherein the execution-instruction signal router is a cross-point switch.

1531. The apparatus of claim 1507, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1532. The apparatus of claim 1507, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1533. The apparatus of claim 1507, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1534. The apparatus of claim 1507, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1535. The apparatus of claim 1531, wherein a sleeping processing resource may be woken by a response to a request.

1536. The apparatus of claim 1535, wherein the response is directed to a specific processing resource.

1537. The apparatus of claim 1535, wherein the response indicates a once locked resource is unlocked.

1538. The apparatus of claim 1507, wherein processing resources are communicatively disposed on a same die.

1539. The apparatus of claim 1538, wherein an execution-instruction signal router is on the same die with processing resources.

1540. The apparatus of claim 1507, wherein the determination is made at a processing cache.

1541. The apparatus of claim 1507, wherein the event variable is set at the processing cache.

1542. The apparatus of claim 1507, wherein the event variable is set in a register.

1543. The apparatus of claim 1507, wherein the event variable is set in a target address in a processing cache.

1544. An apparatus of sharing memory, comprising:

a memory, the memory for storing instructions;

a processing resource that may issue a plurality of processing instructions stored in the memory, wherein the instructions issue signals to:

obtain a storage request including an event address at a processing cache;

combine a parameter with an event variable through in a processing cache;

provide a reply to a processing resource bus from the combination with the event address, which will wake an appropriate processing resource from sleep that is waiting on an event, if the event value is set not to sleep.

1545. The apparatus of claim 1544, wherein the system is completed within a single processing cycle.

1546. The apparatus of claim 1544, wherein the storage request is an execution-instruction signal for execution by a type of processing resource.

1547. The apparatus of claim 1546, further comprising,

route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

1548. The apparatus of claim 1546, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

1549. The apparatus of claim 1546, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

1550. The apparatus of claim 1546, wherein the execution-instruction signal includes an operation-code.

1551. The apparatus of claim 1550, wherein the operation-code indicates a type of resource on which to execute.

1552. The apparatus of claim 1546, wherein the execution-instruction signal includes values from a status register.

1553. The apparatus of claim 1546, wherein the execution-instruction signal includes values from a priority bit status register.

1554. The apparatus of claim 1546, wherein the execution-instruction signal includes a processing resource identifier.

1555. The apparatus of claim 1554, wherein the processing resource identifier is an integer processing unit number.

1556. The apparatus of claim 1546, wherein the execution-instruction signal includes an address.

1557. The apparatus of claim 1544, wherein an other processing resource may be an delegating processing resource.

1558. The apparatus of claim 1544, wherein a processing resource is an integer processing unit.

1559. The apparatus of claim 1544, wherein a processing resource is a mathematical processing unit.

1560. The apparatus of claim 1544, wherein a processing resource is a memory management unit.

1561. The apparatus of claim 1544, wherein a processing resource is a vector processing unit.

1562. The apparatus of claim 1544, wherein a processing resource is a digital signal processing unit.

1563. The apparatus of claim 1544, wherein a processing resource is a graphics processing unit.

1564. The apparatus of claim 1544, wherein a processing resource is an input/output controller processing unit.

1565. The apparatus of claim 1544, wherein a processing resource is an execution-instruction processing cache.

1566. The apparatus of claim 1544, wherein delegation occurs through an execution-instruction signal router.

1567. The apparatus of claim 1566, wherein the execution-instruction signal router is a cross-point switch.

1568. The apparatus of claim 1544, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1569. The apparatus of claim 1544, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1570. The apparatus of claim 1544, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1571. The apparatus of claim 1544, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1572. The apparatus of claim 1568, wherein a sleeping processing resource may be woken by a response to a request.

1573. The apparatus of claim 1572, wherein the response is directed to a specific processing resource.

1574. The apparatus of claim 1572, wherein the response indicates a once locked resource is unlocked.

1575. The apparatus of claim 1544, wherein processing resources are communicatively disposed on a same die.

1576. The apparatus of claim 1575, wherein an execution-instruction signal router is on the same die with processing resources.

1577. The apparatus of claim 1544, wherein the parameter is a bit-mask.

1578. The apparatus of claim 1544, wherein the combination is a logical-OR.

1579. An apparatus of sharing memory, comprising:

a memory, the memory for storing instructions;

a processing resource that may issue a plurality of processing instructions stored in the memory, wherein the instructions issue signals to:

obtain a storage request including a semaphore address at a processing cache;

set a semaphore variable in a processing cache at the semaphore address;

provide an operating system trap-call to wait on a processing queue, if the storage request includes a wait-on-semaphore instruction;

provide an operating system trap-call to signal on a processing queue, if the storage request includes a signal-on-semaphore instruction;

1580. The apparatus of claim 1579, wherein the system is completed within a single processing cycle.

1581. The apparatus of claim 1579, wherein the storage request is an execution-instruction signal for execution by a type of processing resource.

1582. The apparatus of claim 1581, further comprising,

route the execution-instruction signal to an other processing resource, if an operation-code within the execution-instruction signal is for an other processing resource.

1583. The apparatus of claim 1581, wherein the execution-instruction signal is obtained at an execution-instruction signal router.

1584. The apparatus of claim 1581, wherein the execution-instruction signal is provided by a delegating processing resource via instruction-instruction signal router.

1585. The apparatus of claim 1581, wherein the execution-instruction signal includes an operation-code.

1586. The apparatus of claim 1585, wherein the operation-code indicates a type of resource on which to execute.

1587. The apparatus of claim 1581, wherein the execution-instruction signal includes values from a status register.

1588. The apparatus of claim 1581, wherein the execution-instruction signal includes values from a priority bit status register.

1589. The apparatus of claim 1581, wherein the execution-instruction signal includes a processing resource identifier.

1590. The apparatus of claim 1589, wherein the processing resource identifier is an integer processing unit number.

1591. The apparatus of claim 1581, wherein the execution-instruction signal includes an address.

1592. The apparatus of claim 1579, wherein an other processing resource may be an delegating processing resource.

1593. The apparatus of claim 1579, wherein a processing resource is an integer processing unit.

1594. The apparatus of claim 1579, wherein a processing resource is a mathematical processing unit.

1595. The apparatus of claim 1579, wherein a processing resource is a memory management unit.

1596. The apparatus of claim 1579, wherein a processing resource is a vector processing unit.

1597. The apparatus of claim 1579, wherein a processing resource is a digital signal processing unit.

1598. The apparatus of claim 1579, wherein a processing resource is a graphics processing unit.

1599. The apparatus of claim 1579, wherein a processing resource is an input/output controller processing unit.

1600. The apparatus of claim 1579, wherein a processing resource is an execution-instruction processing cache.

1601. The apparatus of claim 1579, wherein delegation occurs through an execution-instruction signal router.

1602. The apparatus of claim 1601, wherein the execution-instruction signal router is a cross-point switch.

1603. The apparatus of claim 1579, wherein a processing resource may sleep while an other processing resource executes delegated execution-instructions.

1604. The apparatus of claim 1579, wherein the execution-instruction signal causes various processing resources dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput.

1605. The apparatus of claim 1579, wherein the execution-instruction signal from processing resources themselves shuts off processing resources while idling.

1606. The apparatus of claim 1579, wherein the execution-instruction signal from processing resources themselves turn on processing resources when execution-instruction signal processing is required.

1607. The apparatus of claim 1603, wherein a sleeping processing resource may be woken by a response to a request.

1608. The apparatus of claim 1607, wherein the response is directed to a specific processing resource.

1609. The apparatus of claim 1607, wherein the response indicates a once locked resource is unlocked.

1610. The apparatus of claim 1579, wherein processing resources are communicatively disposed on a same die.

1611. The apparatus of claim 1610, wherein an execution-instruction signal router is on the same die with processing resources.

1612. The apparatus of claim 1579, wherein the trap-call causes rescheduling so that other processes can run on a particular processing resource.

1613. The apparatus of claim 1579, wherein other processes include threads.

1614. The apparatus of claim 1579, wherein other processes include threads.

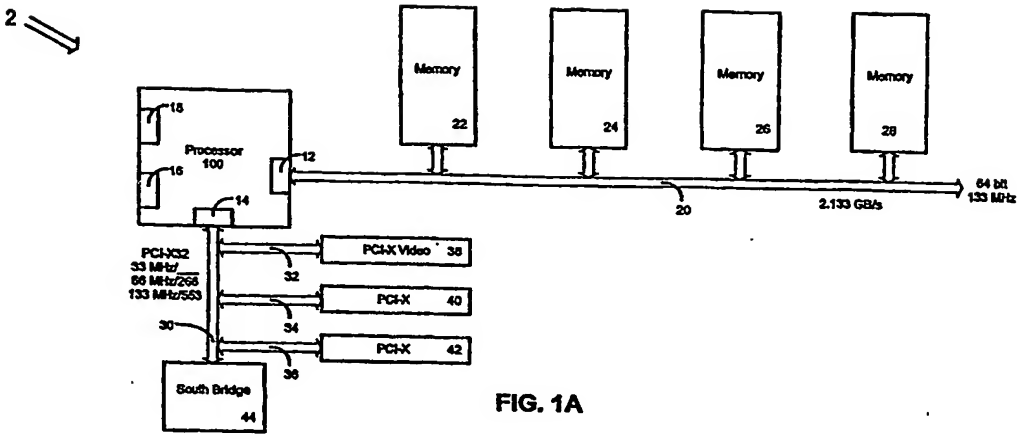

2
Memory
22
Memory
24
Memory
26
Memory
28
18
Processor
100
16
12
14
20
2.133 GB/s
64 bit
133 MHz
PCI-X32
33 MHz/
66 MHz/266
133 MHz/533
30
32
PCI-X Video
38
34
PCI-X
40
36
PCI-X
42
South Bridge
44

FIG. 1A

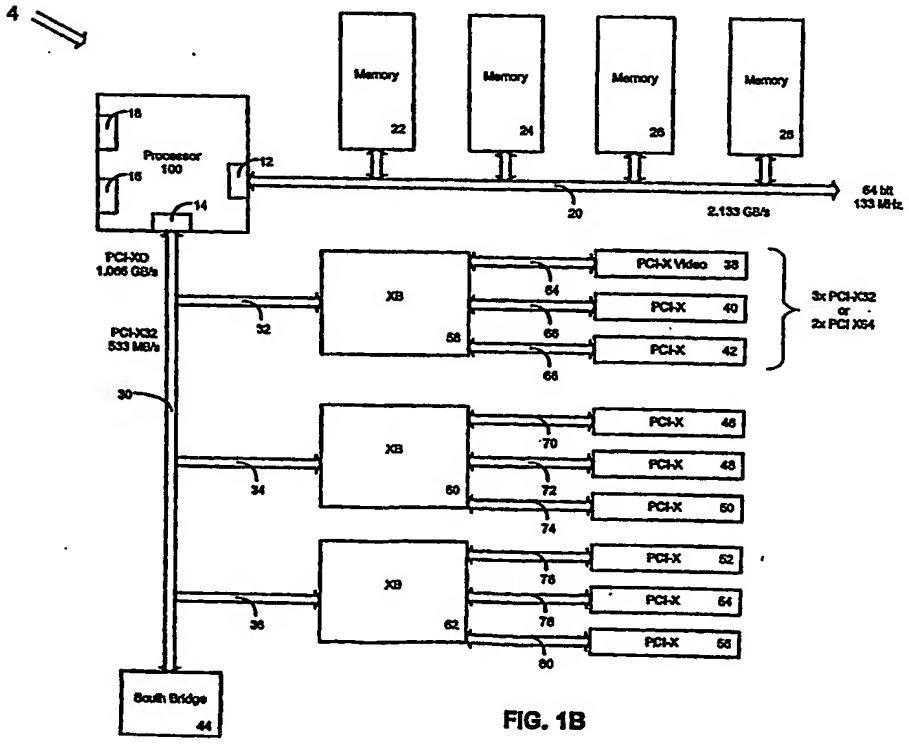

4
Memory
22
Memory
24
Memory
26
Memory
28
18
Processor
100
16
12
14
20
2.133 GB/s
64 bit
133 MHz
PCI-X0
1.066 GB/s
32
XB
58
64
66
68
PCI-X Video
38
PCI-X
40
PCI-X
42
3x PCI-X32
or
2x PCI X64
PCI-X32
533 MB/s
30
34
XB
60
70
72
74
PCI-X
46
PCI-X
48
PCI-X
50
36
XB
62
76
78
80
PCI-X
52
PCI-X
54
PCI-X
56
South Bridge
44

FIG. 1B

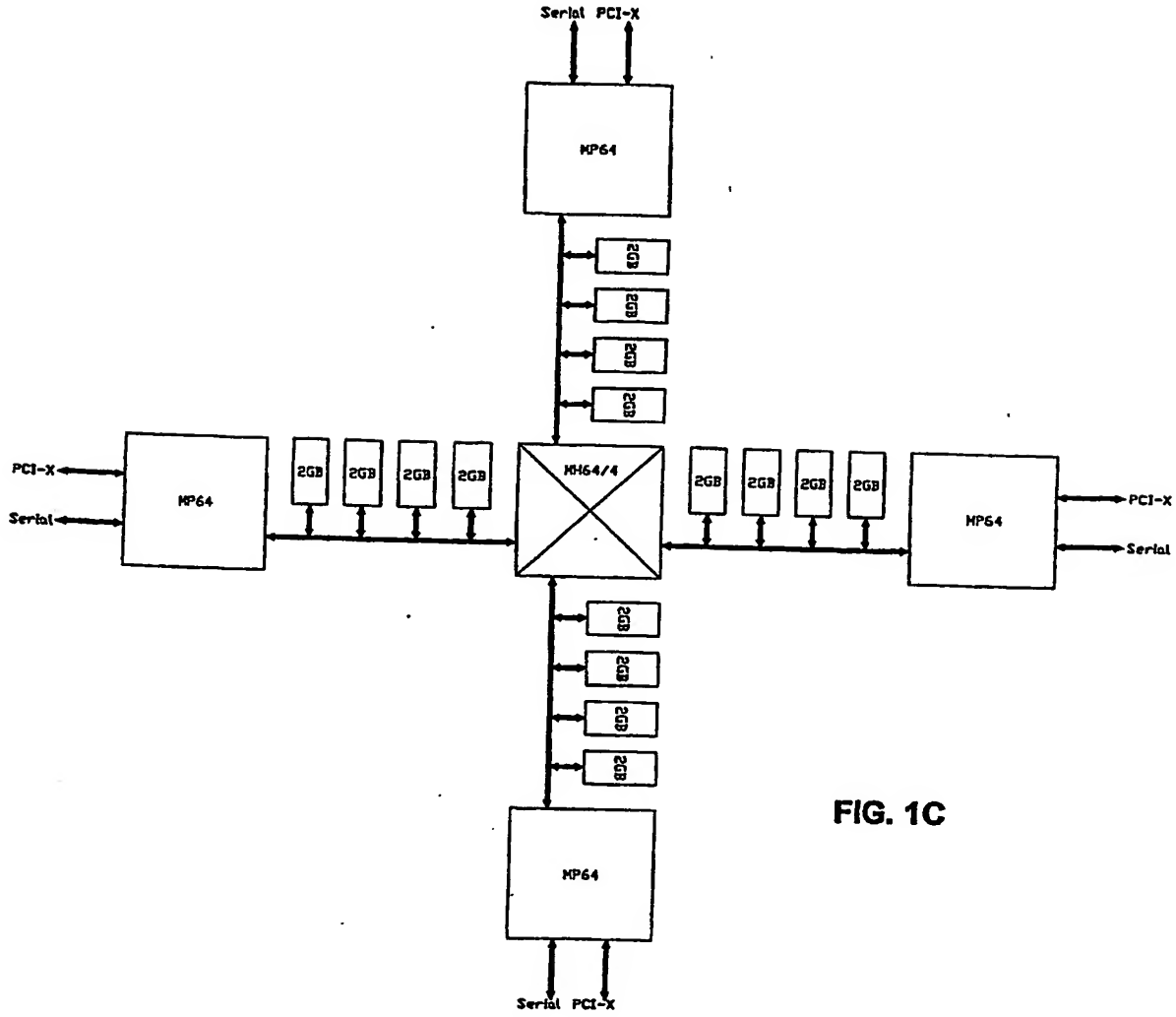

Serial
PCI-X
MP64
2GB
2GB
2GB
2GB
PCI-X
Serial
MP64
2GB
2GB
2GB
2GB
MH64/4
2GB
2GB
2GB
2GB
MP64
PCI-X
Serial
2GB
2GB
2GB
2GB
MP64
Serial
PCI-X

FIG. 1C

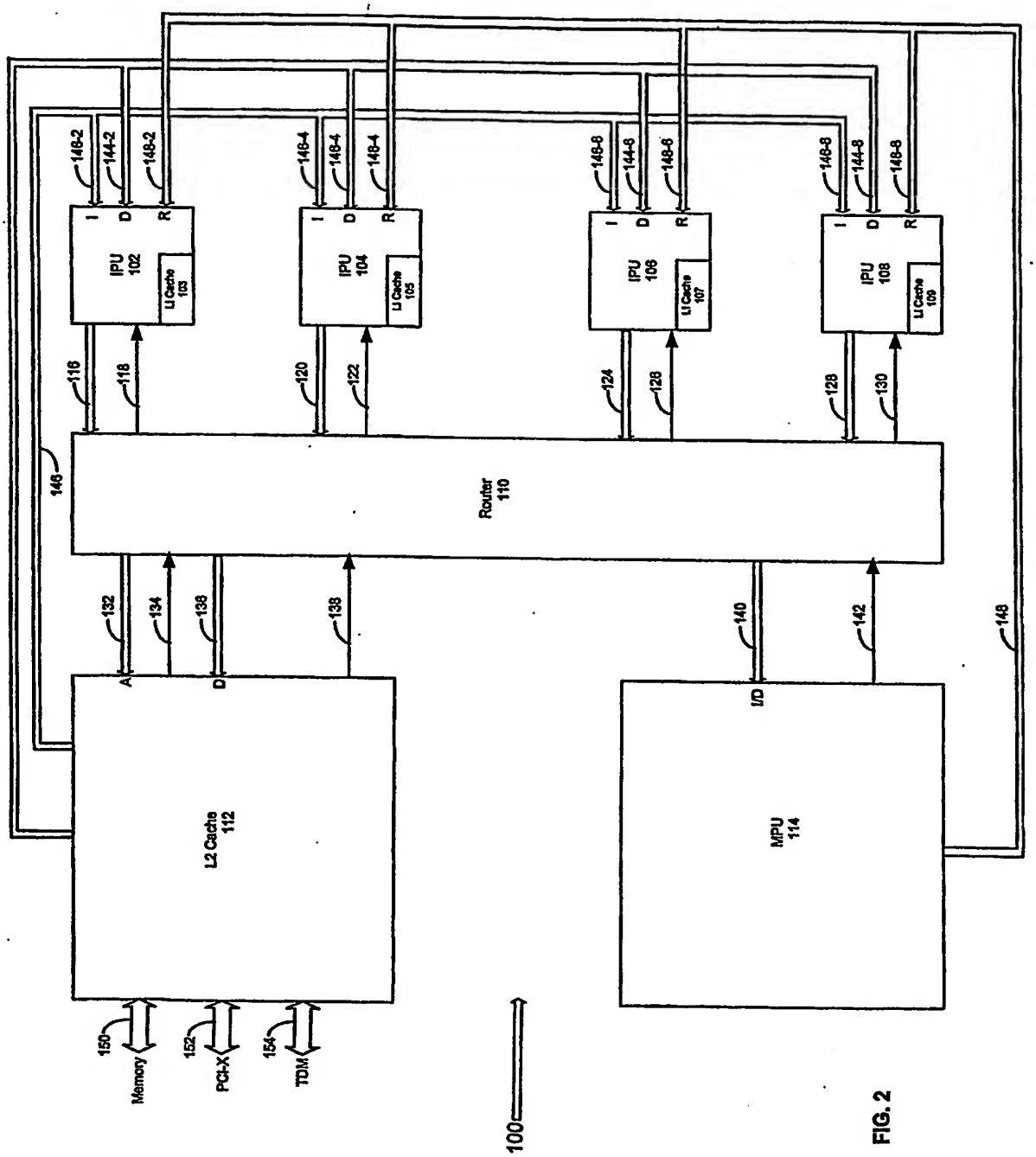

IPU 102
IPU 104
IPU 106
IPU 108
L1 Cache 103
L1 Cache 105
L1 Cache 107
L1 Cache 109
I
D
R
Router 110
L2 Cache 112
MPU 114
A
D
I/D
Memory
PCI-X
TDM
100

FIG. 2

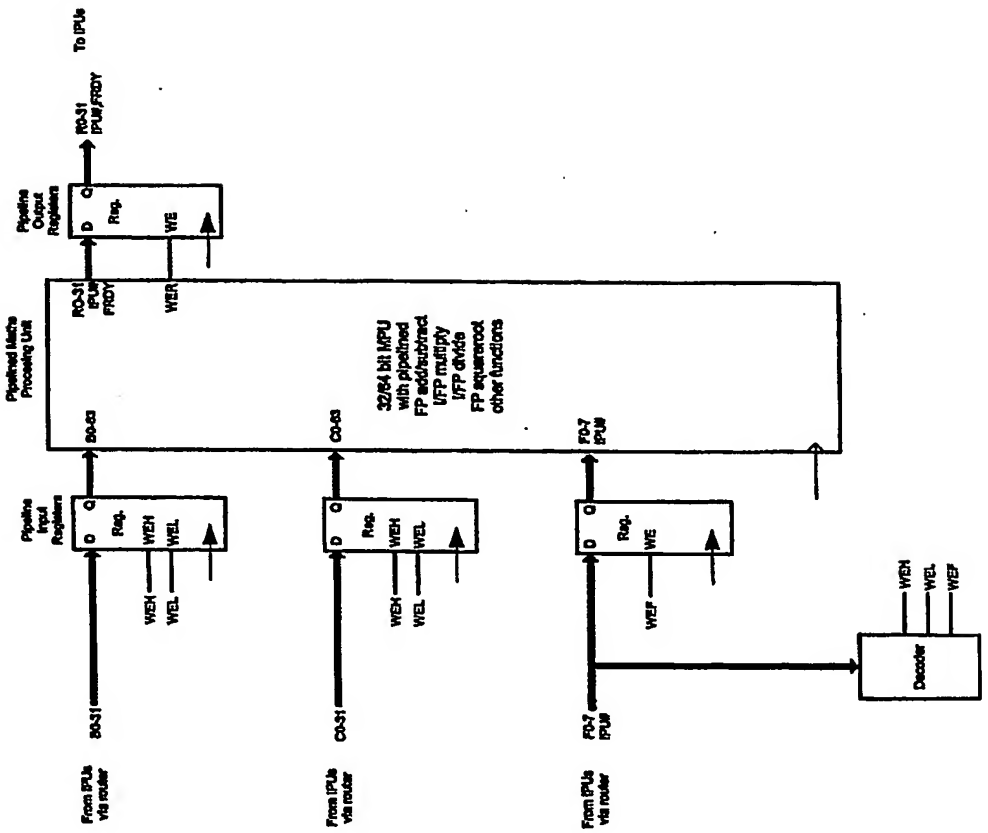

To IPUs
R0-31 IPUR/FRDY
Pipeline Output Registers
D
Q
Reg.
WE
WER
Pipelined Matrix Processing Unit
32/64 bit MPU with pipelined FP add/subtract, I/FP multiply, I/FP divide, FP squareroot, other functions
B0-31
C0-31
F0-7 IPUR
Pipeline Input Registers
WEH
WEL
WEF
Decoder
From IPUs via router

FIG. 3

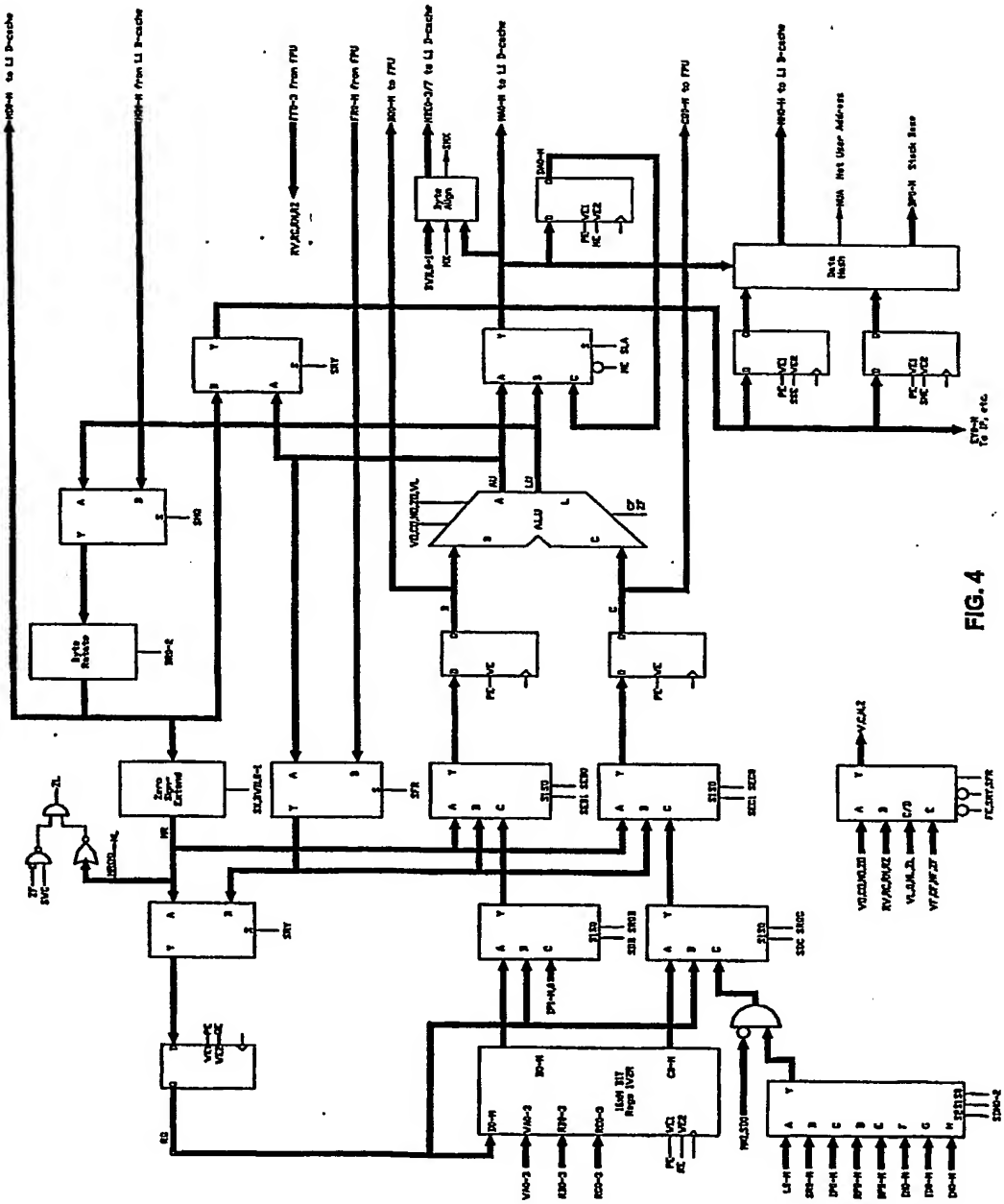


FIG. 4

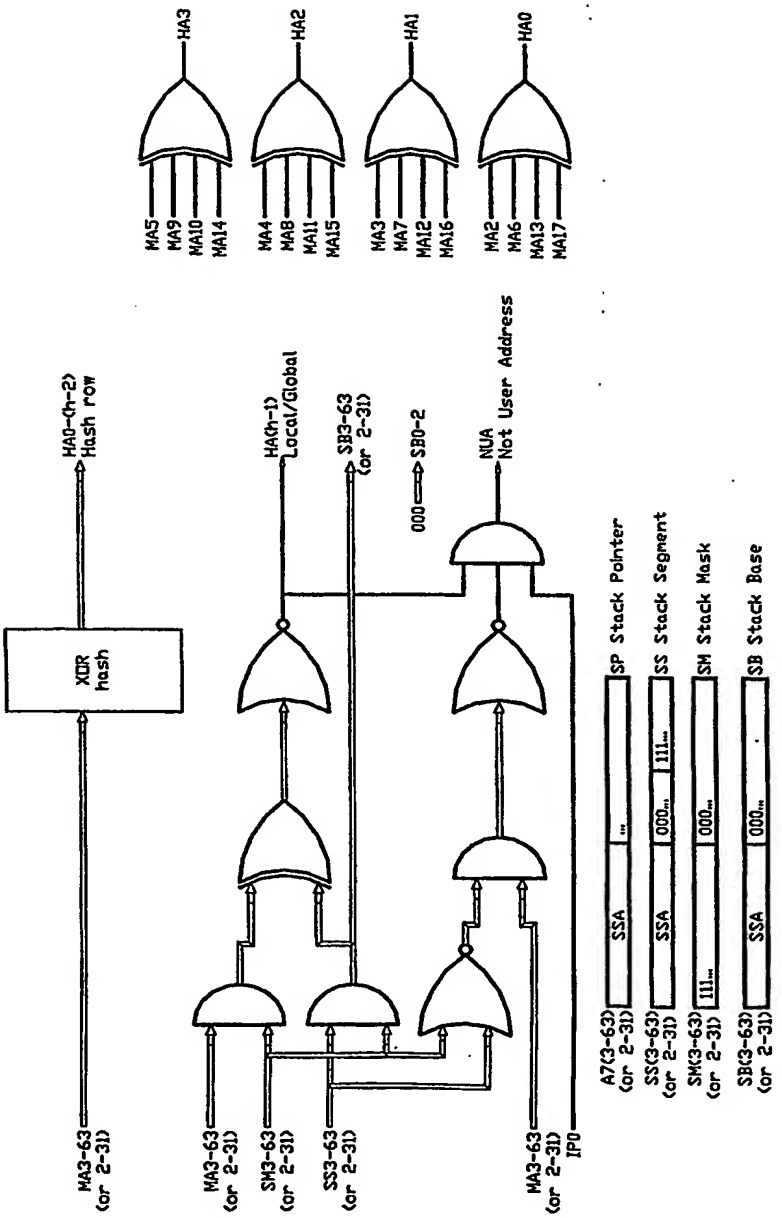

HA3
HA2
HA1
HA0
MA5
MA9
MA10
MA14
MA4
MA8
MA11
MA15
MA3
MA7
MA12
MA16
MA2
MA6
MA13
MA17
HA0-(n-2)
Hash row
HA(n-1)
Local/Global
SB3-63
(or 2-31)
000 ⟹ SB0-2
NUA
Not User Address
XOR
hash
MA3-63
(or 2-31)
MA3-63
(or 2-31)
SM3-63
(or 2-31)
SS3-63
(or 2-31)
MA3-63
(or 2-31)
IP0
A7(3-63)
(or 2-31)
SSA
SP Stack Pointer
SS(3-63)
(or 2-31)
SSA
000...
111...
SS Stack Segment
SM(3-63)
(or 2-31)
111...
000...
SM Stack Mask
SB(3-63)
(or 2-31)
SSA
000...
SB Stack Base

FIG. 5

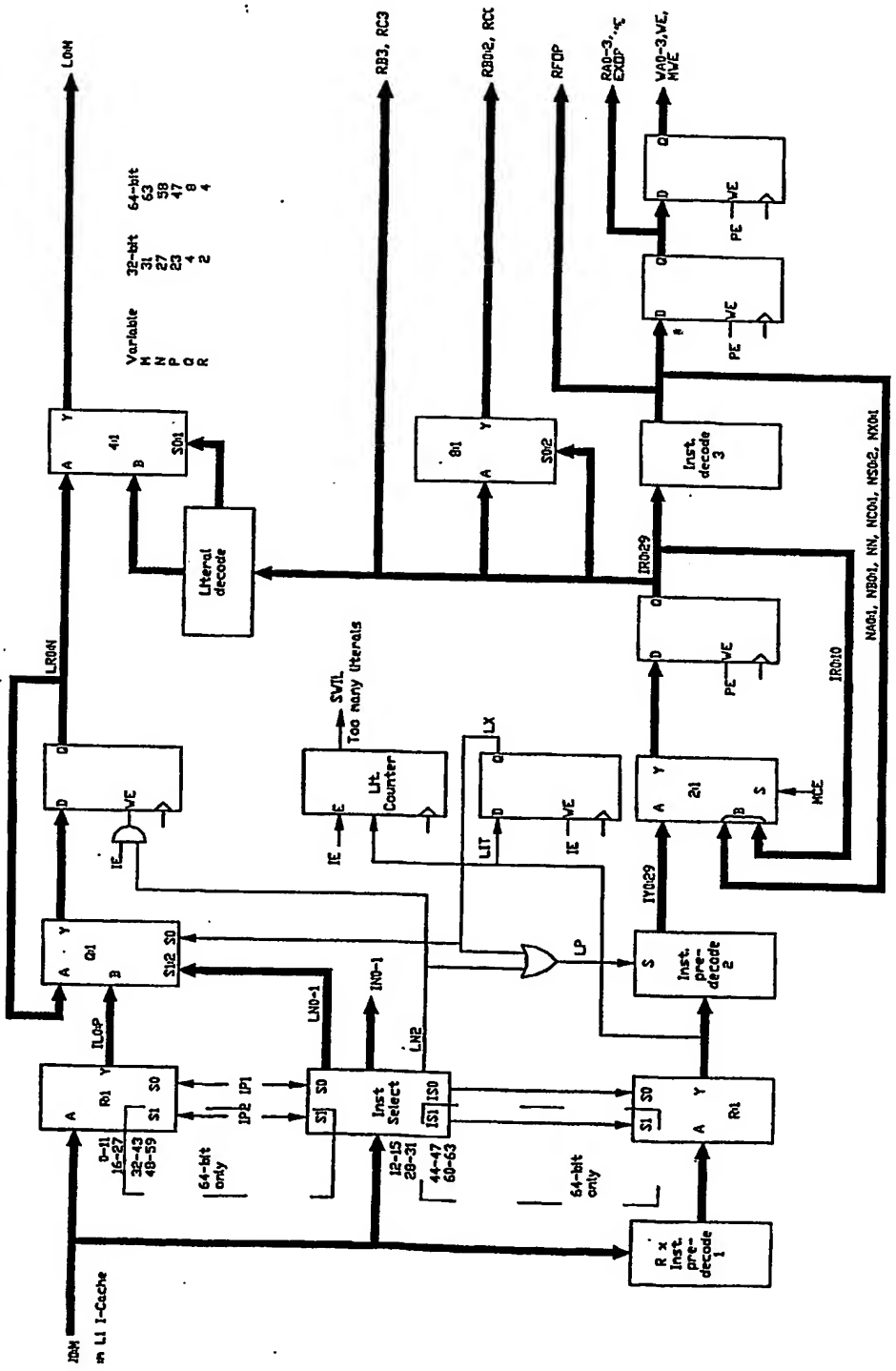

LOM
RB3, RC3
RB02, RC1
RFOP
RA0-3, EXOP
VA0-3, VE, MVE
Variable
M
N
P
Q
R
32-bit
31
27
23
4
2
64-bit
63
58
47
8
4
Literal decode
Inst. decode 3
Inst. pre-decode 2
Inst Select
R x Inst. pre-decode 1
Lit. Counter
SVIL
Too many literals
LR04
IR029
IR010
IY029
LP
LX
LIT
IE
ILOP
LN0-1
LN1
LN2
IP1
IP2
IS0
IS1
0-11
16-27
32-43
48-59
12-15
28-31
44-47
60-63
64-bit only
64-bit only
IDM
m L1 I-Cache

FIG. 6

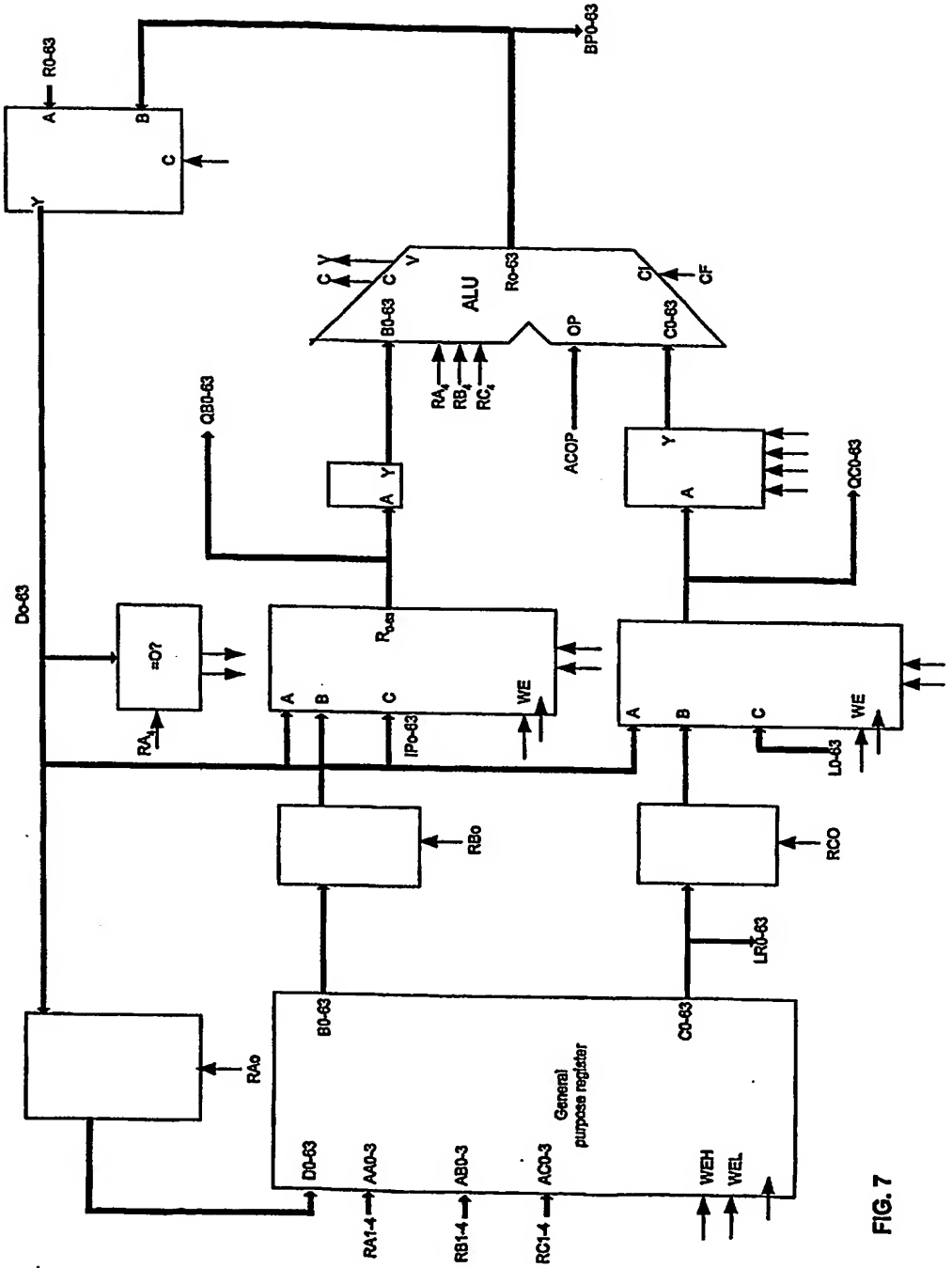

R0-63
A
B
C
Y
BP0-63
ALU
Ro-63
CI
CF
B0-63
C0-63
OP
RA
RB
RC
ACOP
QB0-63
QC0-63
Do-63
=0?
WE
IPo-63
LA-63
RBo
RCo
LR0-63
RAo
D0-63
AA0-3
AB0-3
AC0-3
General purpose register
WEH
WEL
RA1-4
RB1-4
RC1-4

FIG. 7

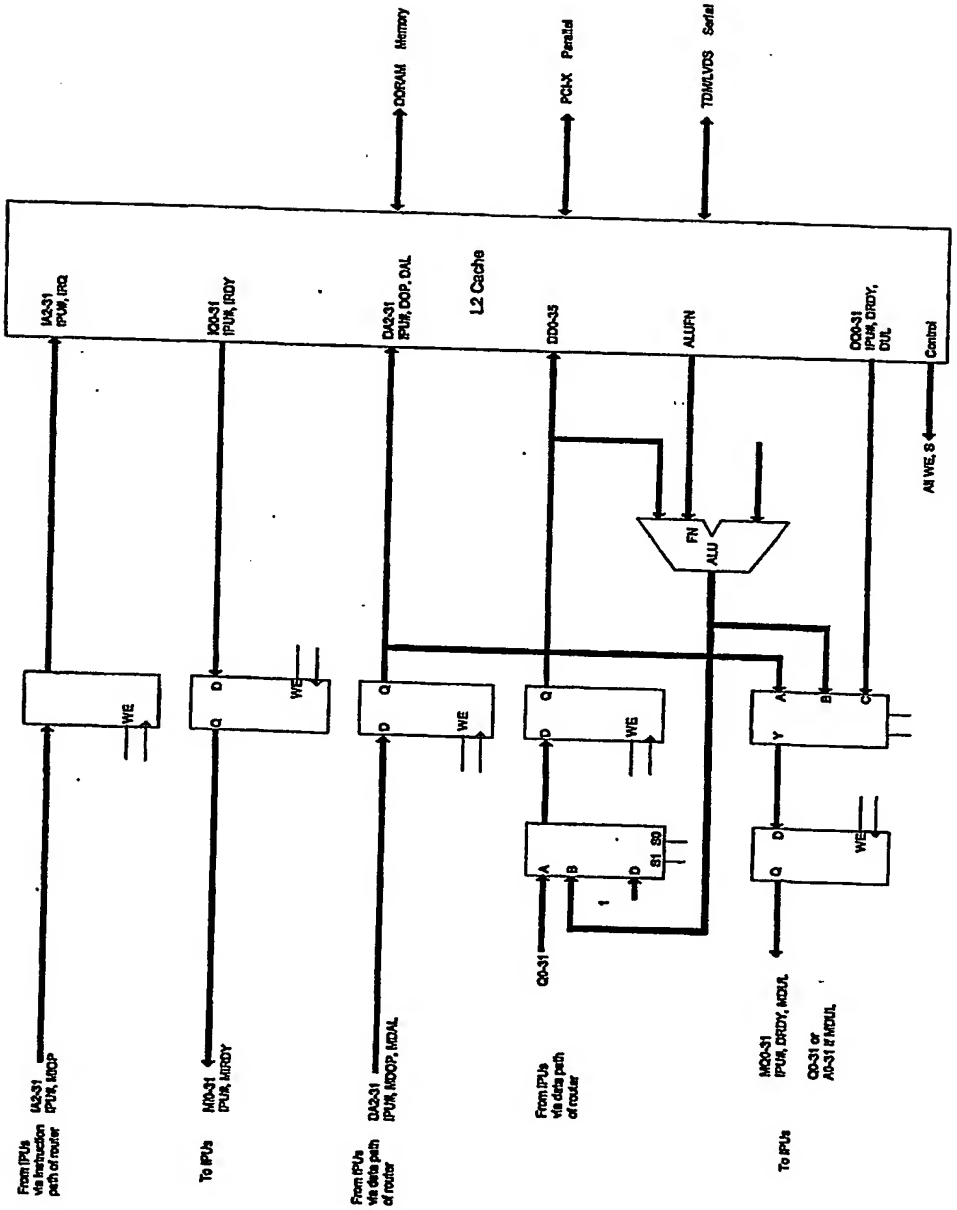

DDRAM Memory
PCI-X Parallel
TDM/LVDS Serial
L2 Cache
IA2-31 IPUR, IRQ
IA0-31 IPUR, IRDY
DA0-31 IPUR, DOP, DAL
DD0-31
ALUFN
DD0-31 IPUR, DRDY, DUL
Control
All WE, S
FN
ALU
WE
D
Q
A
B
C
Y
S0
S1
1
From IPUs via instruction path of router
DA0-31 IPUR, MDOP
MA0-31 IPUR, MIRDY
To IPUs
From IPUs via data path of router
DA0-31 IPUR, MDOP, MDAL
DD-31
From IPUs via data path of router
MD0-31 IPUR, DRDY, MDUL
DD-31 or AD-31 if MDUL
To IPUs

FIG. 8

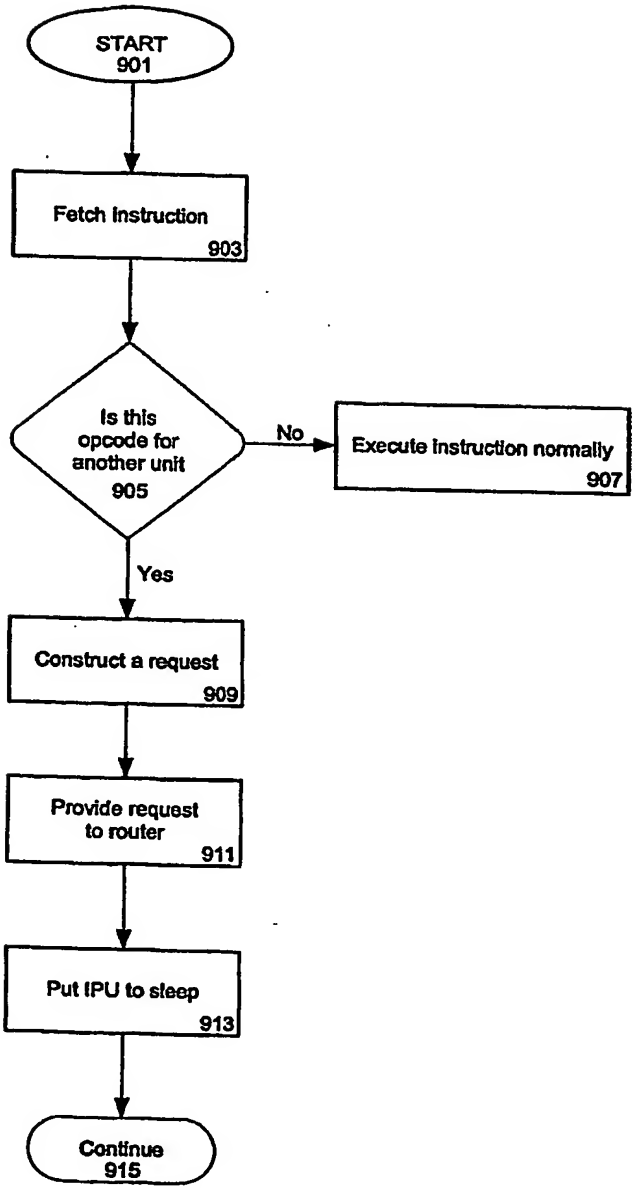

START
901
Fetch Instruction
903
Is this opcode for another unit
905
No
Execute Instruction normally
907
Yes
Construct a request
909
Provide request to router
911
Put IPU to sleep
913
Continue
915

FIG. 9

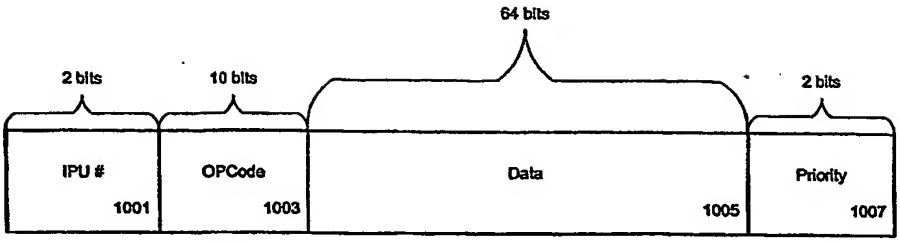

64 bits
2 bits
10 bits
2 bits
IPU #
1001
OPCode
1003
Data
1005
Priority
1007

FIG. 10

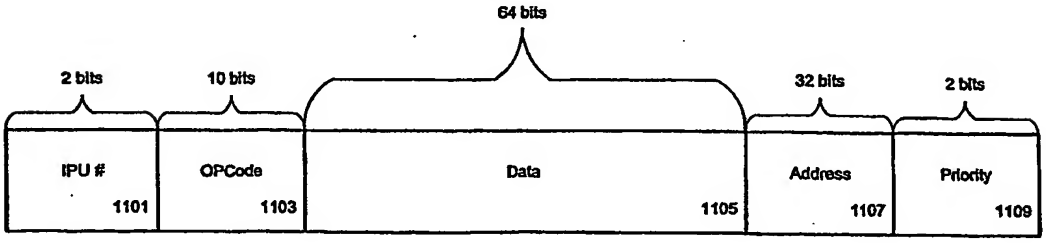

64 bits
2 bits
10 bits
32 bits
2 bits
IPU #
1101
OPCode
1103
Data
1105
Address
1107
Priority
1109

FIG. 11

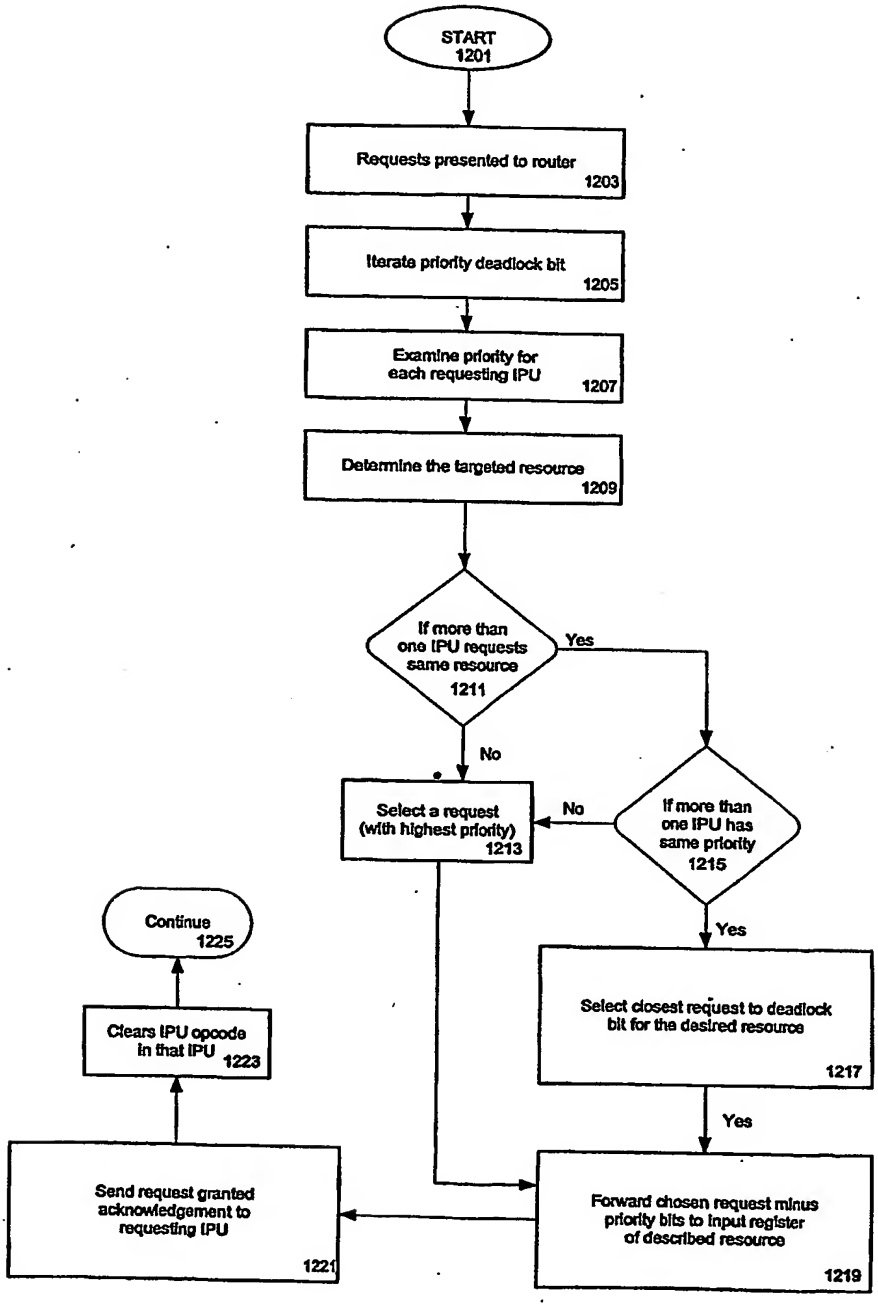

START
1201
Requests presented to router
1203
Iterate priority deadlock bit
1205
Examine priority for each requesting IPU
1207
Determine the targeted resource
1209
If more than one IPU requests same resource
1211
Yes
No
Select a request (with highest priority)
1213
No
If more than one IPU has same priority
1215
Yes
Select closest request to deadlock bit for the desired resource
1217
Yes
Forward chosen request minus priority bits to input register of described resource
1219
Send request granted acknowledgement to requesting IPU
1221
Clears IPU opcode in that IPU
1223
Continue
1225

FIG. 12

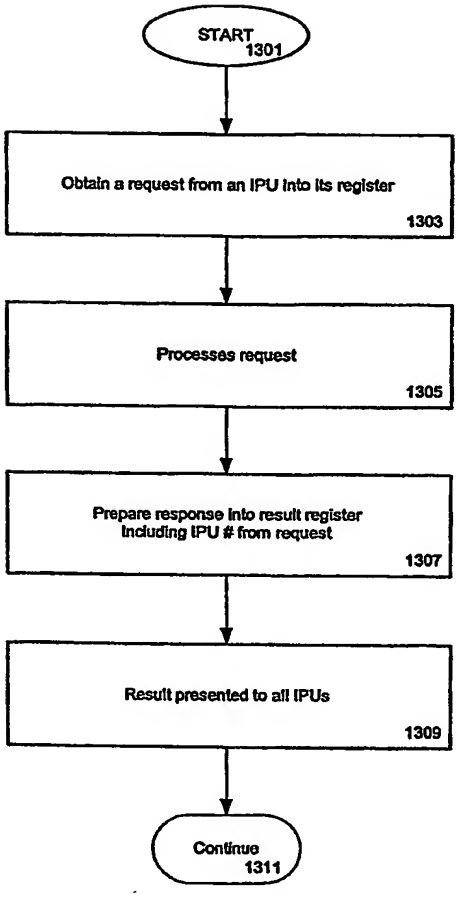

START
1301
Obtain a request from an IPU into its register
1303
Processes request
1305
Prepare response into result register
Including IPU # from request
1307
Result presented to all IPUs
1309
Continue
1311

FIG. 13

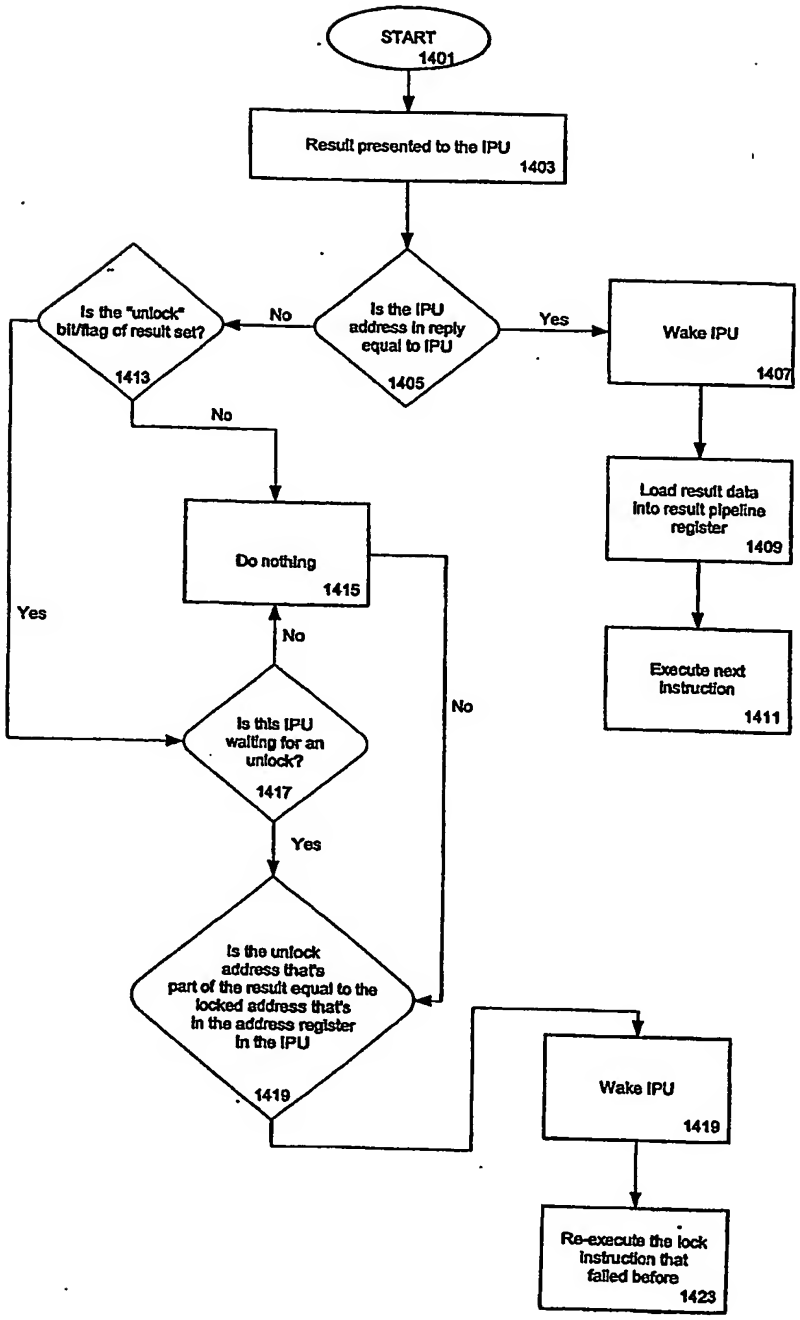

START
1401
Result presented to the IPU
1403
Is the IPU address in reply equal to IPU
1405
No
Yes
Is the "unlock" bit/flag of result set?
1413
Wake IPU
1407
No
Load result data into result pipeline register
1409
Do nothing
1415
Execute next instruction
1411
Yes
No
Is this IPU waiting for an unlock?
1417
No
Yes
Is the unlock address that's part of the result equal to the locked address that's in the address register in the IPU
1419
Wake IPU
1419
Re-execute the lock instruction that failed before
1423

FIG. 14

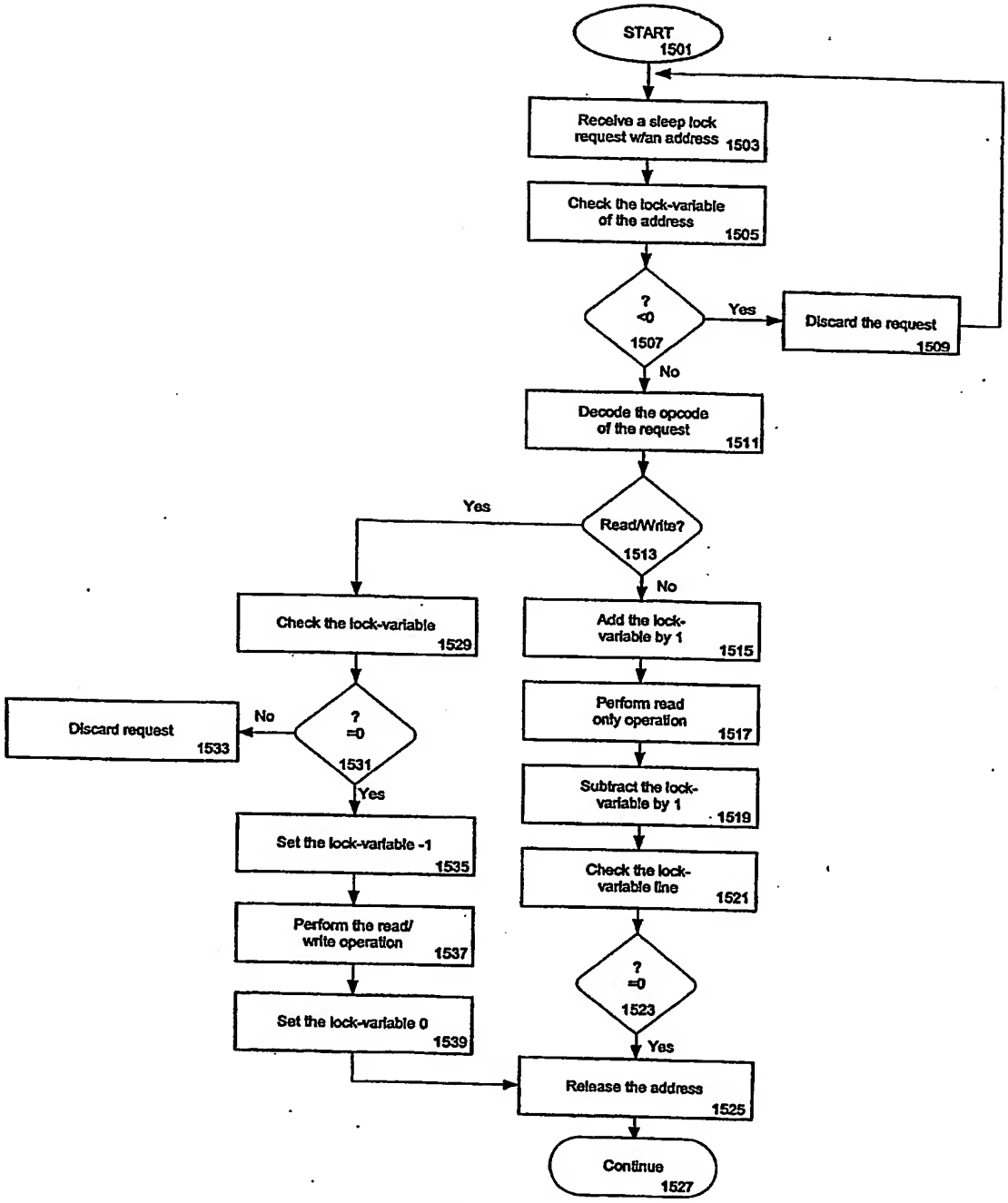

START
1501
Receive a sleep lock request w/an address
1503
Check the lock-variable of the address
1505
? <0
1507
Yes
Discard the request
1509
No
Decode the opcode of the request
1511
Read/Write?
1513
Yes
No
Check the lock-variable
1529
Add the lock-variable by 1
1515
Perform read only operation
1517
Subtract the lock-variable by 1
1519
Check the lock-variable line
1521
? =0
1523
Yes
Release the address
1525
Continue
1527
? =0
1531
No
Discard request
1533
Yes
Set the lock-variable -1
1535
Perform the read/ write operation
1537
Set the lock-variable 0
1539

FIG. 15

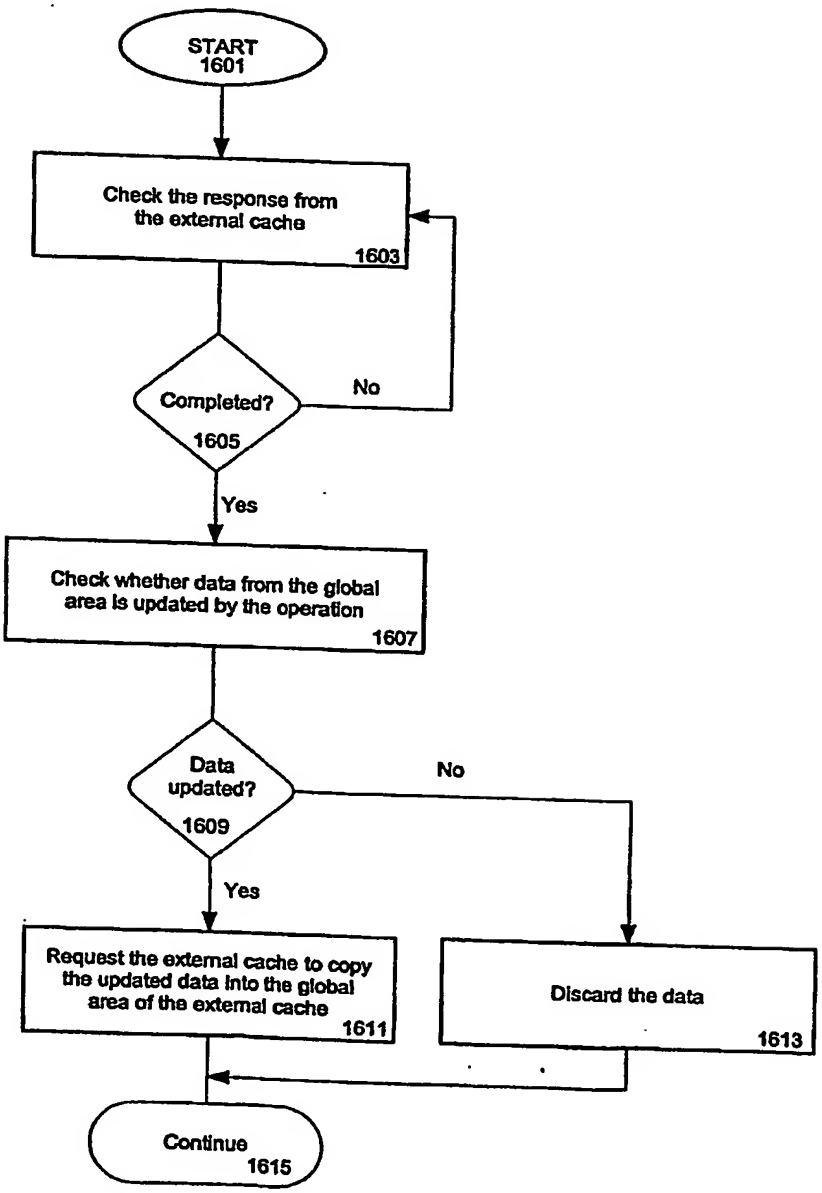

START
1601
Check the response from the external cache
1603
Completed?
1605
No
Yes
Check whether data from the global area is updated by the operation
1607
Data updated?
1609
No
Yes
Request the external cache to copy the updated data into the global area of the external cache
1611
Discard the data
1613
Continue
1615

FIG. 16

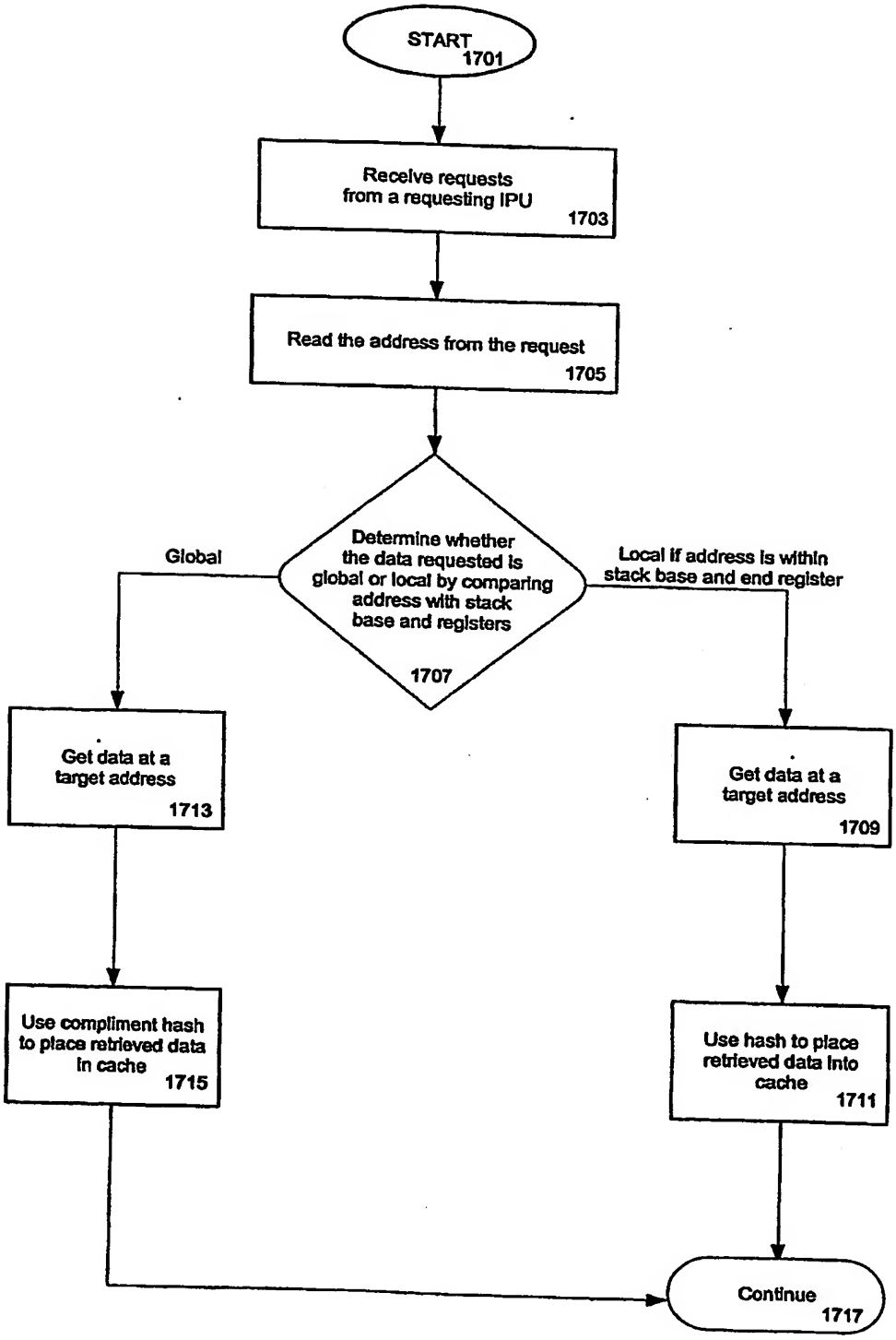

START
1701
Receive requests from a requesting IPU
1703
Read the address from the request
1705
Determine whether the data requested is global or local by comparing address with stack base and registers
1707
Global
Local if address is within stack base and end register
Get data at a target address
1713
Get data at a target address
1709
Use compliment hash to place retrieved data in cache
1715
Use hash to place retrieved data into cache
1711
Continue
1717

FIG. 17

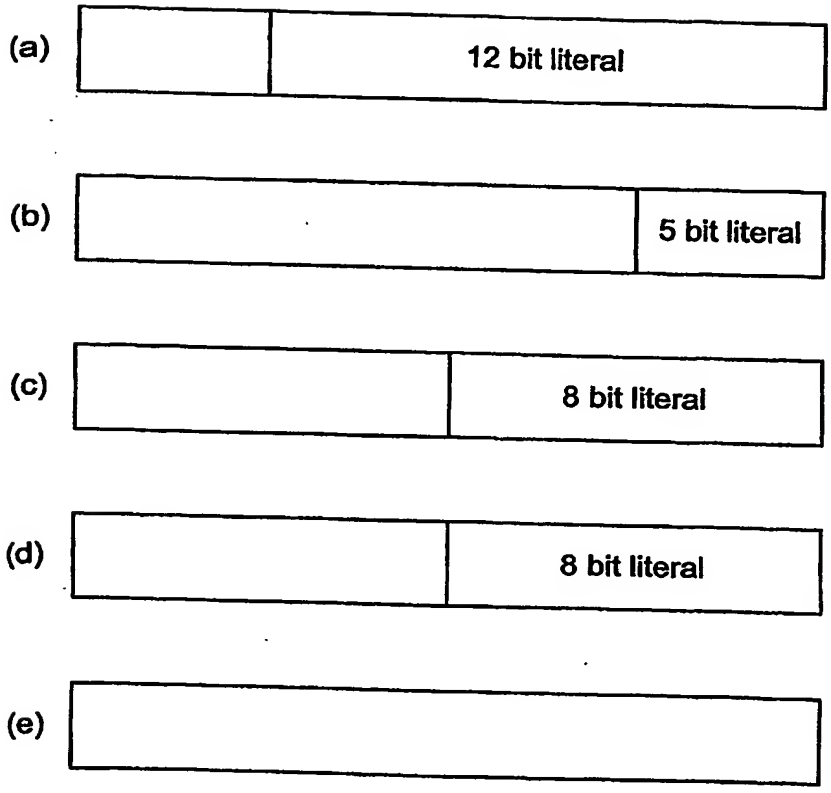

(a)
12 bit literal
(b)
5 bit literal
(c)
8 bit literal
(d)
8 bit literal
(e)

FIG. 18

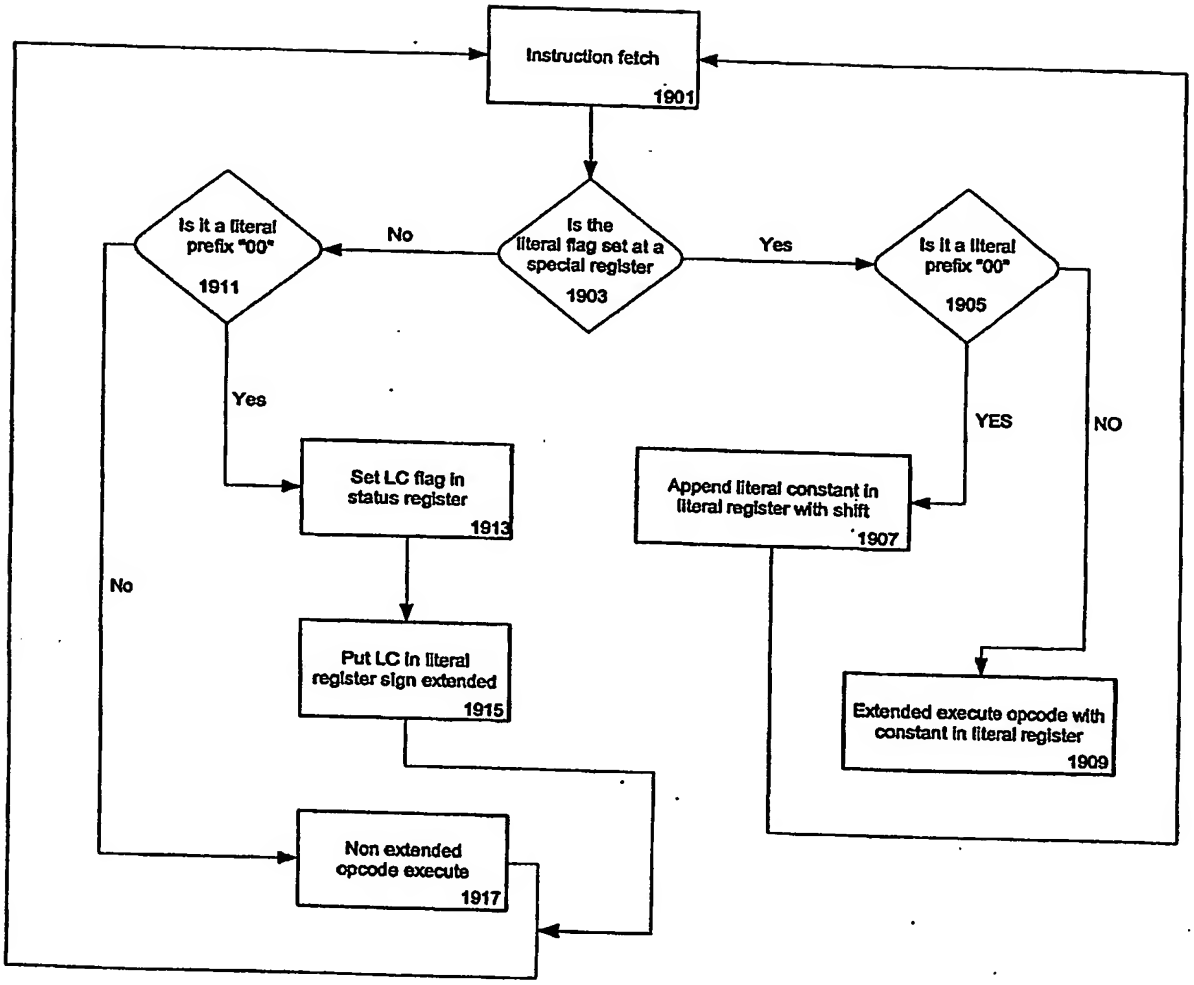

Instruction fetch
1901
Is the literal flag set at a special register
1903
No
Yes
Is it a literal prefix "00"
1911
Is it a literal prefix "00"
1905
Yes
YES
NO
Set LC flag in status register
1913
Append literal constant in literal register with shift
1907
No
Put LC in literal register sign extended
1915
Extended execute opcode with constant in literal register
1909
Non extended opcode execute
1917

FIG. 19

Status Register

Cycle Flags	Operations requesting number of cycles
0 (2101)	Normal (simple cycle) (2109)
1 (2103	Double (two cycles/double processes load/save) (2111)
2 (2105)	Extend (multi cycles/double processes load/save) (2113)
3 (2107)	Repeated (multiple cycles/single/double process status) (2115)

FIG. 21

Type Flag	Number of Registers	Number of Bits	Type
0 (2201)	8 (2205)	32 (2209)	Data (2213)
1 (2203)	8 (2207)	32 (2211)	Address (2215)

FIG. 22

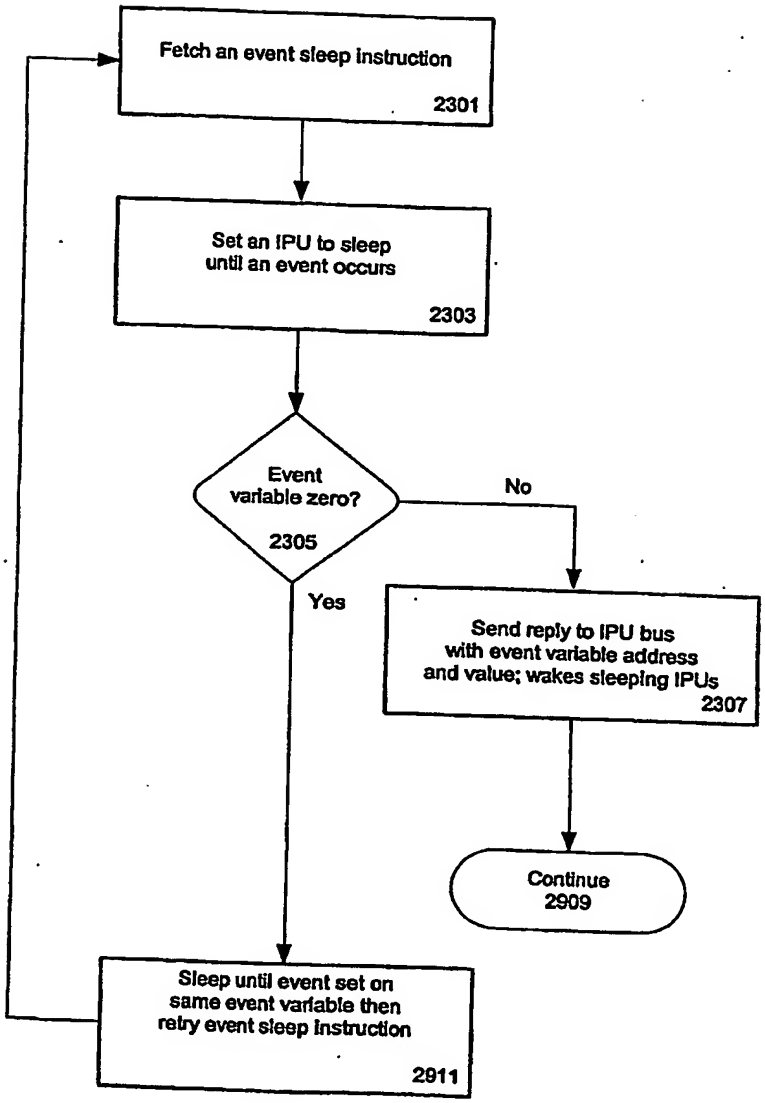

Fetch an event sleep instruction
2301
Set an IPU to sleep until an event occurs
2303
Event variable zero?
2305
No
Yes
Send reply to IPU bus with event variable address and value; wakes sleeping IPUs
2307
Continue
2909
Sleep until event set on same event variable then retry event sleep instruction
2911

FIG. 23

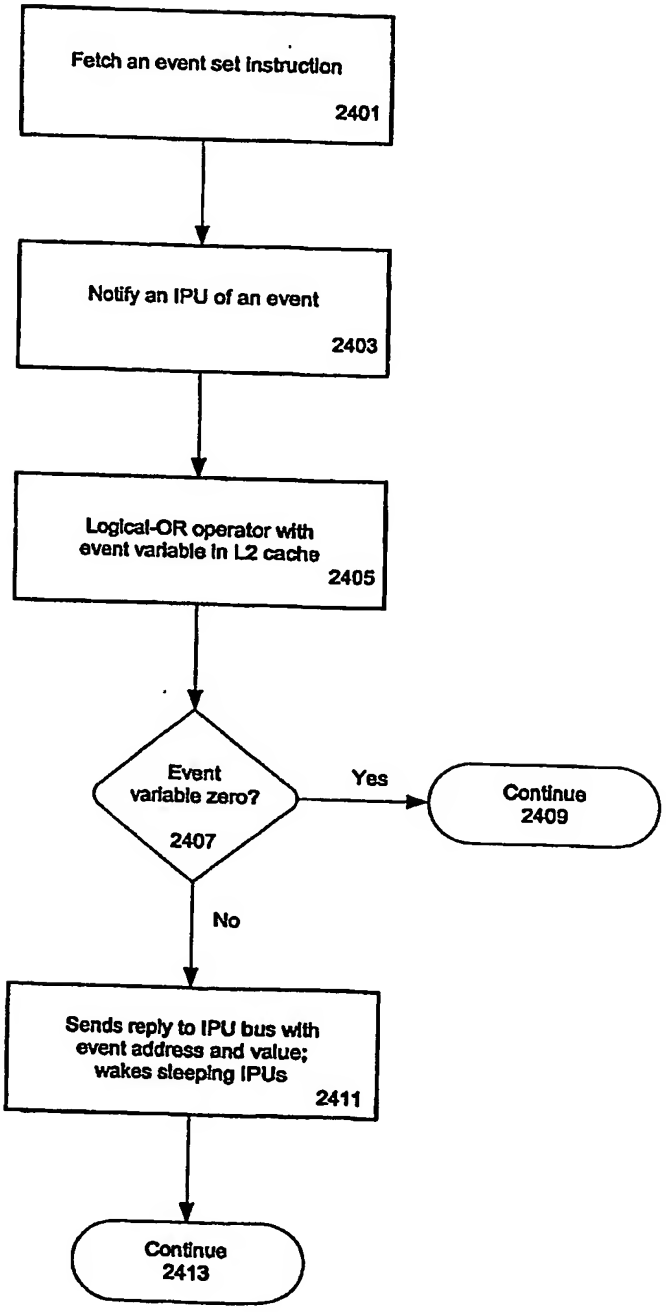

Fetch an event set instruction
2401
Notify an IPU of an event
2403
Logical-OR operator with event variable in L2 cache
2405
Event variable zero?
2407
Yes
Continue
2409
No
Sends reply to IPU bus with event address and value; wakes sleeping IPUs
2411
Continue
2413

FIG. 24

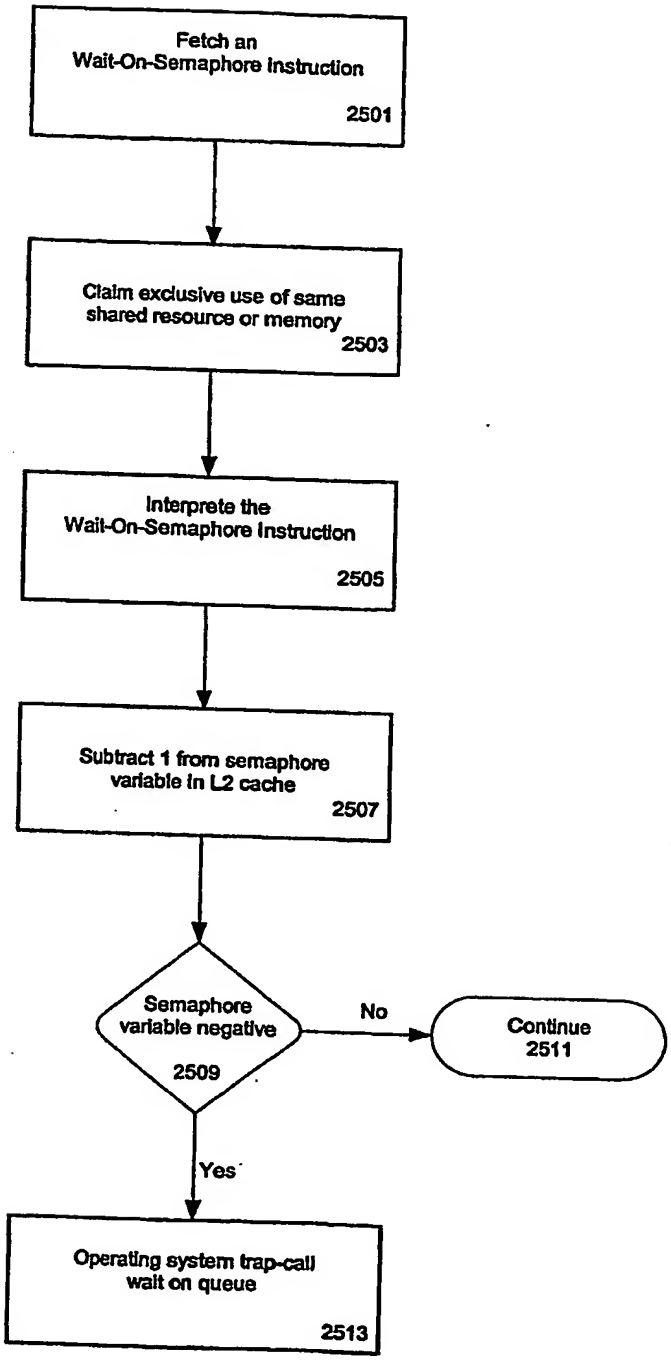

Fetch an
Wait-On-Semaphore Instruction
2501
Claim exclusive use of same
shared resource or memory
2503
Interprete the
Wait-On-Semaphore Instruction
2505
Subtract 1 from semaphore
variable in L2 cache
2507
Semaphore
variable negative
2509
No
Continue
2511
Yes
Operating system trap-call
wait on queue
2513

FIG. 25

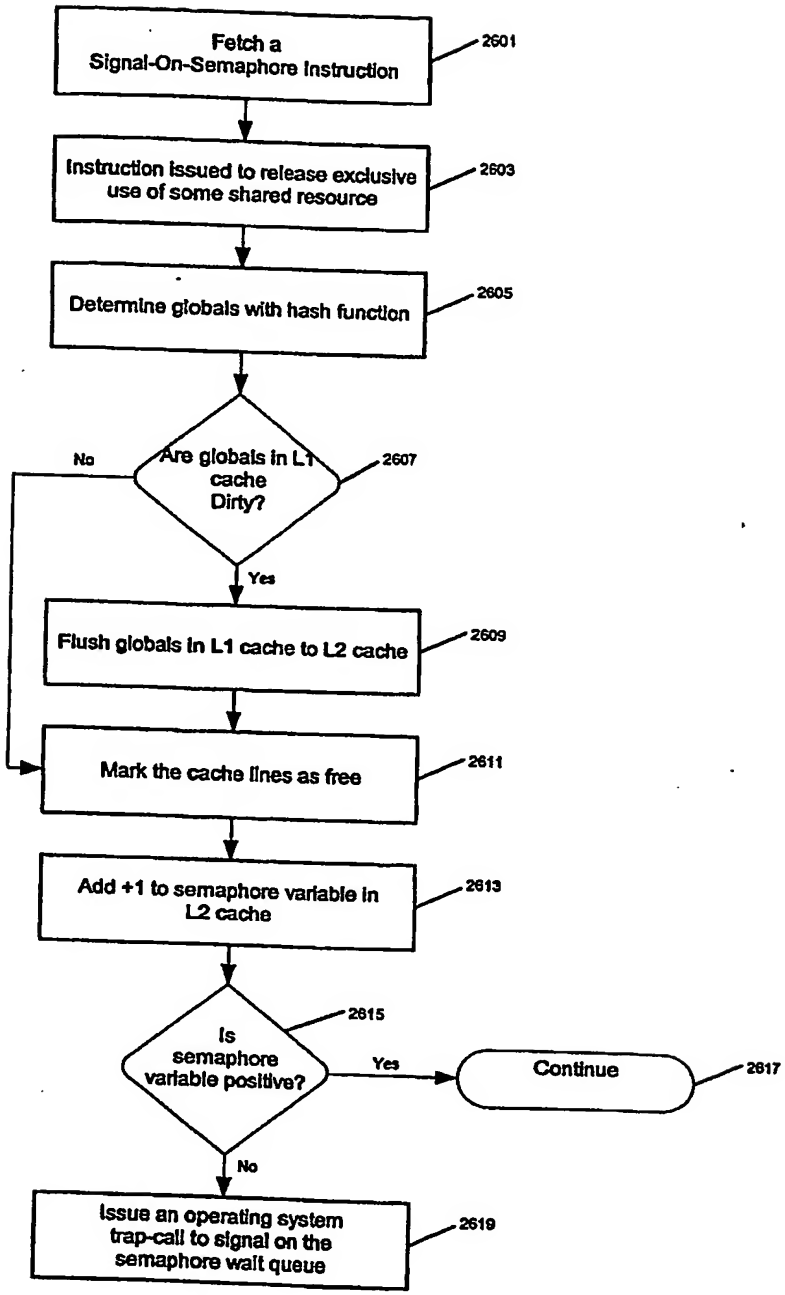

Fetch a Signal-On-Semaphore Instruction
2601
Instruction issued to release exclusive use of some shared resource
2603
Determine globals with hash function
2605
Are globals in L1 cache Dirty?
2607
No
Yes
Flush globals in L1 cache to L2 cache
2609
Mark the cache lines as free
2611
Add +1 to semaphore variable in L2 cache
2613
Is semaphore variable positive?
2615
Yes
Continue
2617
No
Issue an operating system trap-call to signal on the semaphore wait queue
2619

FIG. 26

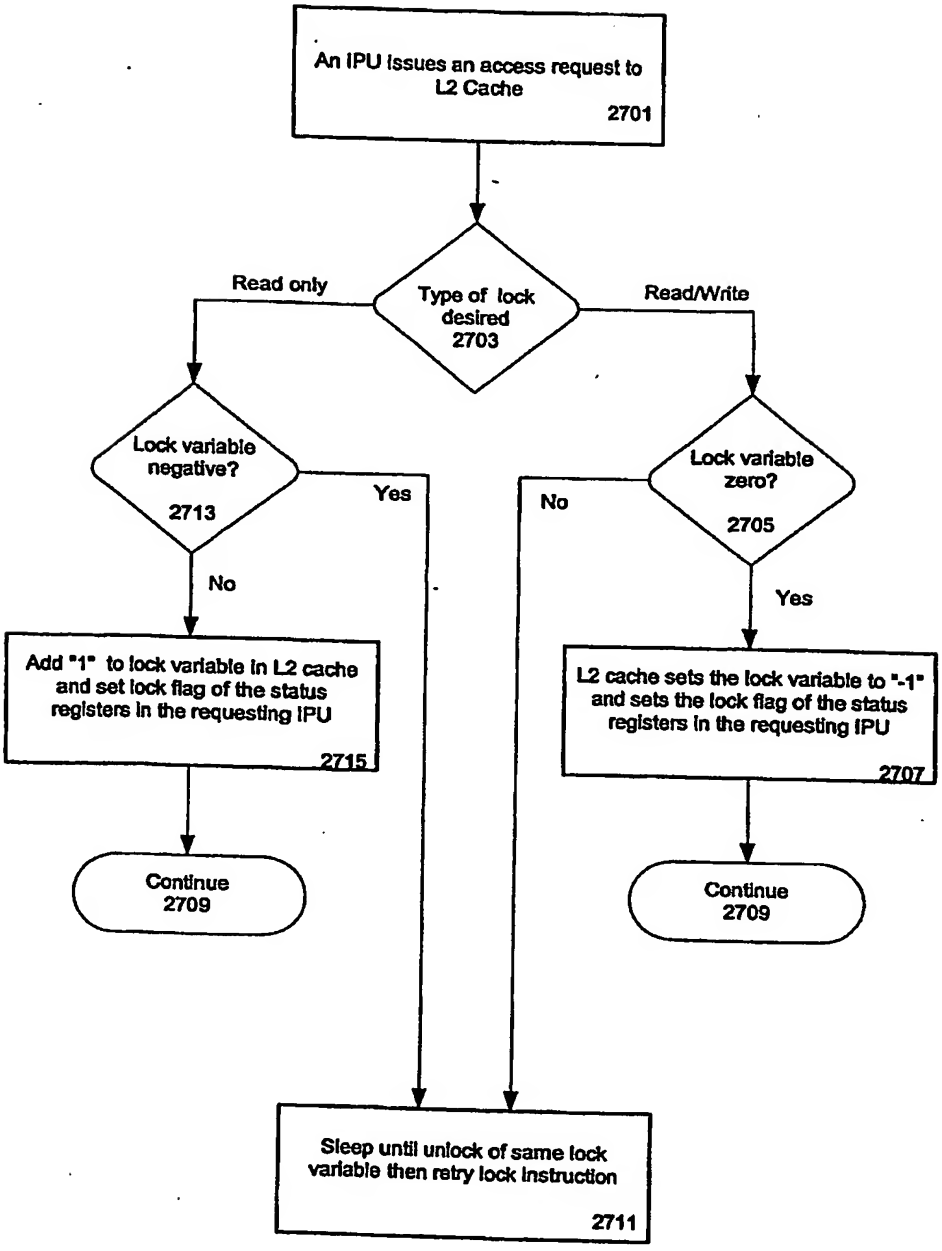

An IPU issues an access request to L2 Cache
2701
Type of lock desired
2703
Read only
Read/Write
Lock variable negative?
2713
Yes
No
Lock variable zero?
2705
No
Yes
Add "1" to lock variable in L2 cache and set lock flag of the status registers in the requesting IPU
2715
L2 cache sets the lock variable to "-1" and sets the lock flag of the status registers in the requesting IPU
2707
Continue
2709
Continue
2709
Sleep until unlock of same lock variable then retry lock instruction
2711

FIG. 27

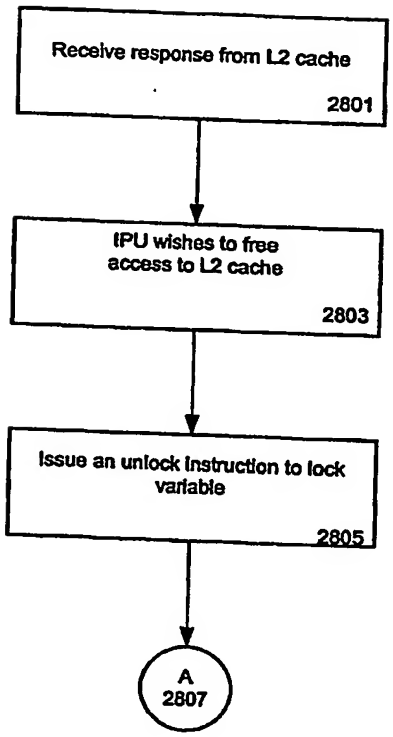

Receive response from L2 cache
2801
IPU wishes to free access to L2 cache
2803
Issue an unlock instruction to lock variable
2805
A
2807

FIG. 28

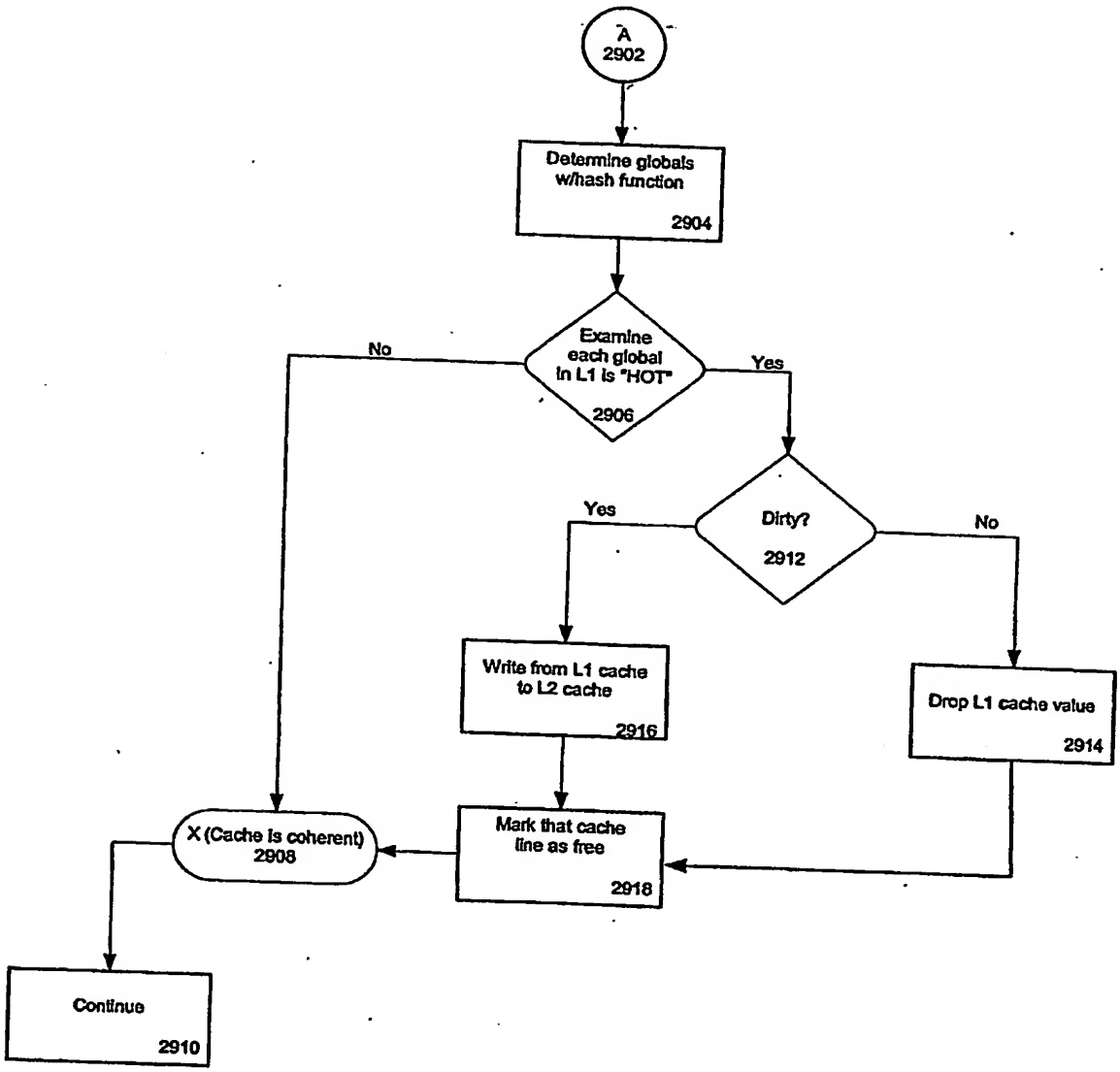

A
2902
Determine globals w/hash function
2904
Examine each global in L1 is "HOT"
2906
No
Yes
Dirty?
2912
Yes
No
Write from L1 cache to L2 cache
2916
Drop L1 cache value
2914
Mark that cache line as free
2918
X (Cache is coherent)
2908
Continue
2910

FIG. 29

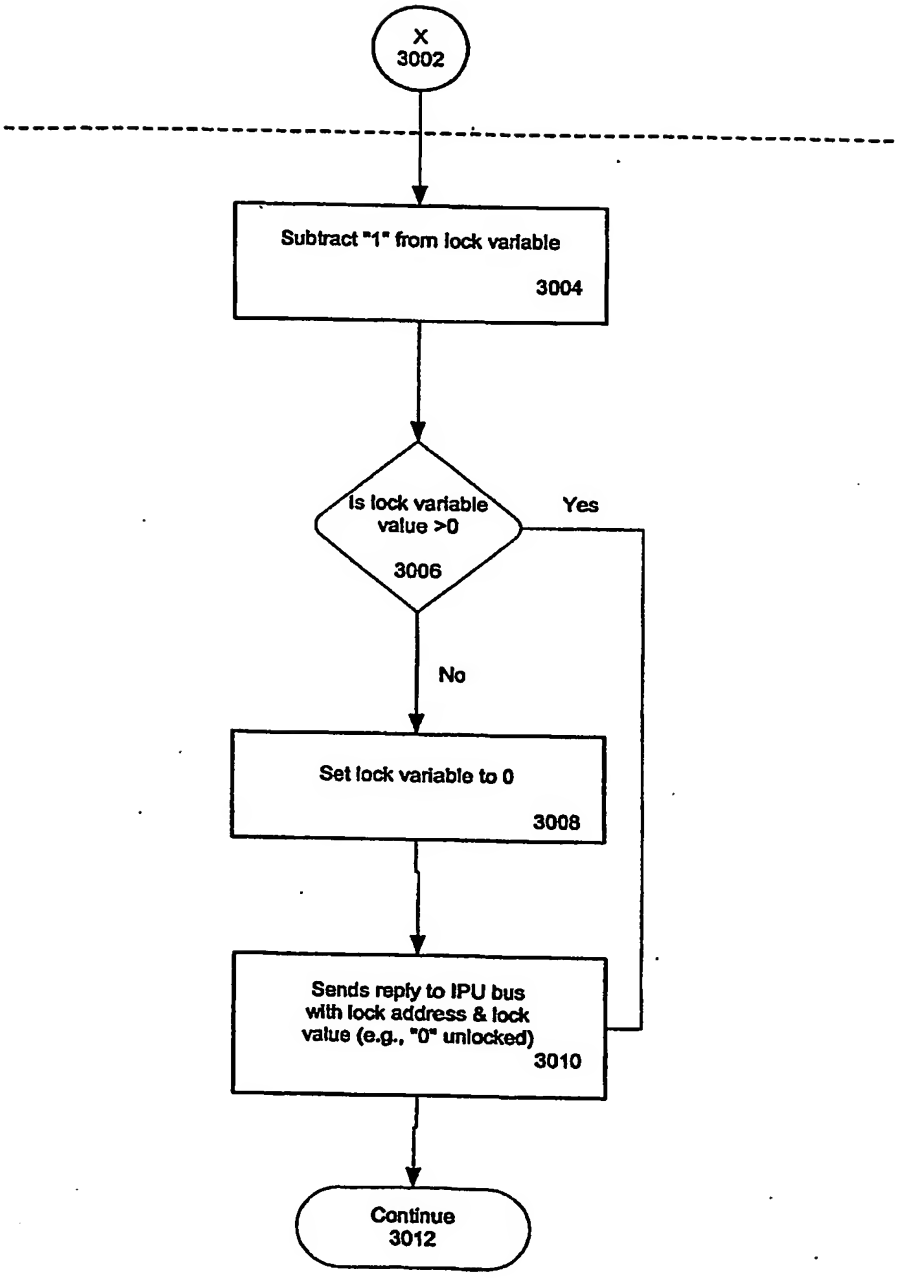

X
3002
Subtract "1" from lock variable
3004
Is lock variable value >0
3006
Yes
No
Set lock variable to 0
3008
Sends reply to IPU bus with lock address & lock value (e.g., "0" unlocked)
3010
Continue
3012

FIG. 30

INTERNATIONAL SEARCH REPORT

International application No.
PCT/AU03/00994

A. CLASSIFICATION OF SUBJECT MATTER

Int. Cl. [7]: G06F 9/50, 12/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
USPTO Web Patent Database, Esp@cenet, WPAT "multiprocessors, memory, cache, share, sleep etc."

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5522083 A (GOVE et al.) 28 May 1996 Abstract, column 36 line 25 and column 36 lines 12 to 16 for example.	1-16,122-143, 621-642,1120-1141
X	EP 794492 A (COMPAQ COMPUTER CORPORATION) 10 September 1997 Column 3 line 49 to column 4 line 27 for example.	144-172,643-671, 1142-1170
X	US 5159689 A (SHIRAISHI) 27 October 1992 Entire document.	1-16

[X] Further documents are listed in the continuation of Box C [X] See patent family annex

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
4 December 2003	17 DEC 2003
Name and mailing address of the ISA/AU AUSTRALIAN PATENT OFFICE PO BOX 200, WODEN ACT 2606, AUSTRALIA E-mail address: pct@ipaustralia.gov.au Facsimile No. (02) 6285 3929	Authorized officer P. THONG Telephone No : (02) 6283 2128

INTERNATIONAL SEARCH REPORT

International application No.
PCT/AU03/00994

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 4346435 A (WISE) 24 August 1982 Entire document.	1-16
X	EP 385136 B1 (NEC CORPORATION) 21 January 1998 Entire document.	1-16
X	US 6035374 A (PANWAR et al.) 7 March 2000 Column 7 lines 1 to 3 and column 8 lines 3 to 7 for example.	101-110
A	US 5860158 A (PAI et al.) 12 January 1999 Entire document.	1-16, 101-110,121-172 620-671,1119-1170
A	US 5210828 A (BOLAN et al.) 11 May 1993 Entire document.	1-16, 101-110,121-172 620-671,1119-1170

INTERNATIONAL SEARCH REPORT

International application No.
PCT/AU03/00994

Box I **Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)**

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos :

because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos :

because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos :

because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a)

Box II **Observations where unity of invention is lacking (Continuation of item 3 of first sheet)**

This International Searching Authority found multiple inventions in this international application, as follows:

See Supplemental Boxes 1 and 2.

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☒ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

1-16, 101-110, 121-172, 620-671 and 1119-1170

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☒ No protest accompanied the payment of additional search fees.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/AU03/00994

Supplemental Box 1
(To be used when the space in any of Boxes I to VIII is not sufficient)

Continuation of Box No: II

The international application does not comply with the requirements of unity of invention because it does not relate to one invention or to a group of inventions so linked as to form a single general inventive concept. In coming to this conclusion the International Searching Authority has found that there are different inventions as follows:

1. Claims 1,122,144,621,643,1120,1142 and any relevant dependent claim(s).
   Method, system or apparatus which involves the delegating of execution-instructions between processing resources via an instruction execution router.

2. Claim 17 and any relevant dependent claim(s).
   Apparatus comprising memory that includes an instruction unit which memory may be accessed simultaneously by processing resources wherein the processing resources and memory are on the same die.

3. Claim 36 and and any relevant dependent claim(s).
   Apparatus comprising a memory cache disposed in communication with an instruction determination unit and an instruction execution unit wherein both units store values into the memory within a single cycle.

4. Claim 55 and any relevant dependent claim(s).
   Apparatus comprising an arbitration unit that includes a priority bit comparator wherein the unit is communicatively disposed with processing resources.

5. Claim 75 and any relevant dependent claim(s).
   A medium whose execution-instruction signals involve a processing resource identifier that identifies the origin of an execution signal and an operation code that identifies a target processing resource

6. Claim 85 and any relevant dependent claim(s).
   Processor with an integer processing unit, math processing unit and a router that interfaces between the units wherein the router is adapted to route the request from the integer processing unit to the math processing unit.

7. Claim 101 and any relevant dependent claim(s).
   Processor with processing units and a cache unit wherein each of the processing units is configured to sleep after sending an access request to the cache unit.

8. Claim 111 and any relevant dependent claim(s).
   Processor with internal and external cache units wherein the internal and external cache units is divided into instruction and data areas and the data areas are further subdivided into local and global areas.

9. Claims 121,620,1119 and any relevant dependent claim(s).
   Method, system or apparatus that involves setting processing resources waiting on shared and locked memory being accessed by other processing resources to sleep until the memory is unlocked.

10. Claims 173,672,1171 and any relevant dependent claim(s).
    Method, system or apparatus that involves determining a priority dead-lock-avoidance value wherein the value is used to select among multiple requests with equal priority

NOTE: See continuation in Supplemental Box 2.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/AU03/00994

Supplemental Box 2
(To be used when the space in any of Boxes I to VIII is not sufficient)

Continuation of Box No: II

11. Claims 211,710,1209 and any relevant dependent claim(s).
Method, system or apparatus that involves preparing a response into a result register wherein the response includes a requesting processing resource identifier and presenting the response to all processing resources.

12. Claims 244,743,1242 and any relevant dependent claim(s).

Method, system or apparatus that involves waking a requesting processing resource if a requesting processing resource identifier identifies the instant processing resource and waking a processing resource if a processing resource is waiting to be unlocked.

13. Claims 278,777,1276 and any relevant dependent claim(s).
Method, system or apparatus that involves comparing a target memory address with register values wherein register values are used to establish a data type of the target memory address for subsequent storage in an apportioned region of cache memory.

14. Claims 319,818,1317 and any relevant dependent claim(s).
Method, system or apparatus that involves a literal constant and a literal prefix relating to a reduced size execution of execution-instructions.

15. Claims 360,859,1358 and any relevant dependent claim(s).
Method, system or apparatus that involves replacing odd address value in an instruction register with an even address found in a binding name table.

16. Claims 118,395,617,894,1116,1393 and any relevant dependent claim(s).
Method, system or apparatus that involves a status register as well as addressing a register specified by a register address or cycle flags.

17. Claims 431,930,1429 and any relevant dependent claim(s).
Method, system or apparatus that involves requesting that a requesting processing resource sleep until it is unlocked if a target memory is locked

18. Claims 468,967,1466 and any relevant dependent claim(s).
Method, system or apparatus that involves determining value types by using a hash function as well as updating value types in a primary cache memory of a processing resource to a secondary cache memory for each type value, if each value type in a primary cache has not been updated to a secondary cache memories.

19. Claims 508,509,546,1008,1045,1507,1544 and any relevant dependent claim(s).
Method, system or apparatus that involves an event variable and the waking of a processing resource from sleep that is waiting on an event if the event value is set not to sleep.

20. Claims 581,1080,1579 and any relevant dependent claim(s).
Method, system or apparatus that involves wait-on semaphore instruction, signal-on semaphore instruction and operating system trap-call.

Since the abovementioned groups of claims do not appear to share any technical features, a "technical relationship" between the inventions, as defined in PCT rule 13.2 does not exist. Accordingly the international application does not relate to one invention or to a single inventive concept, a priori.

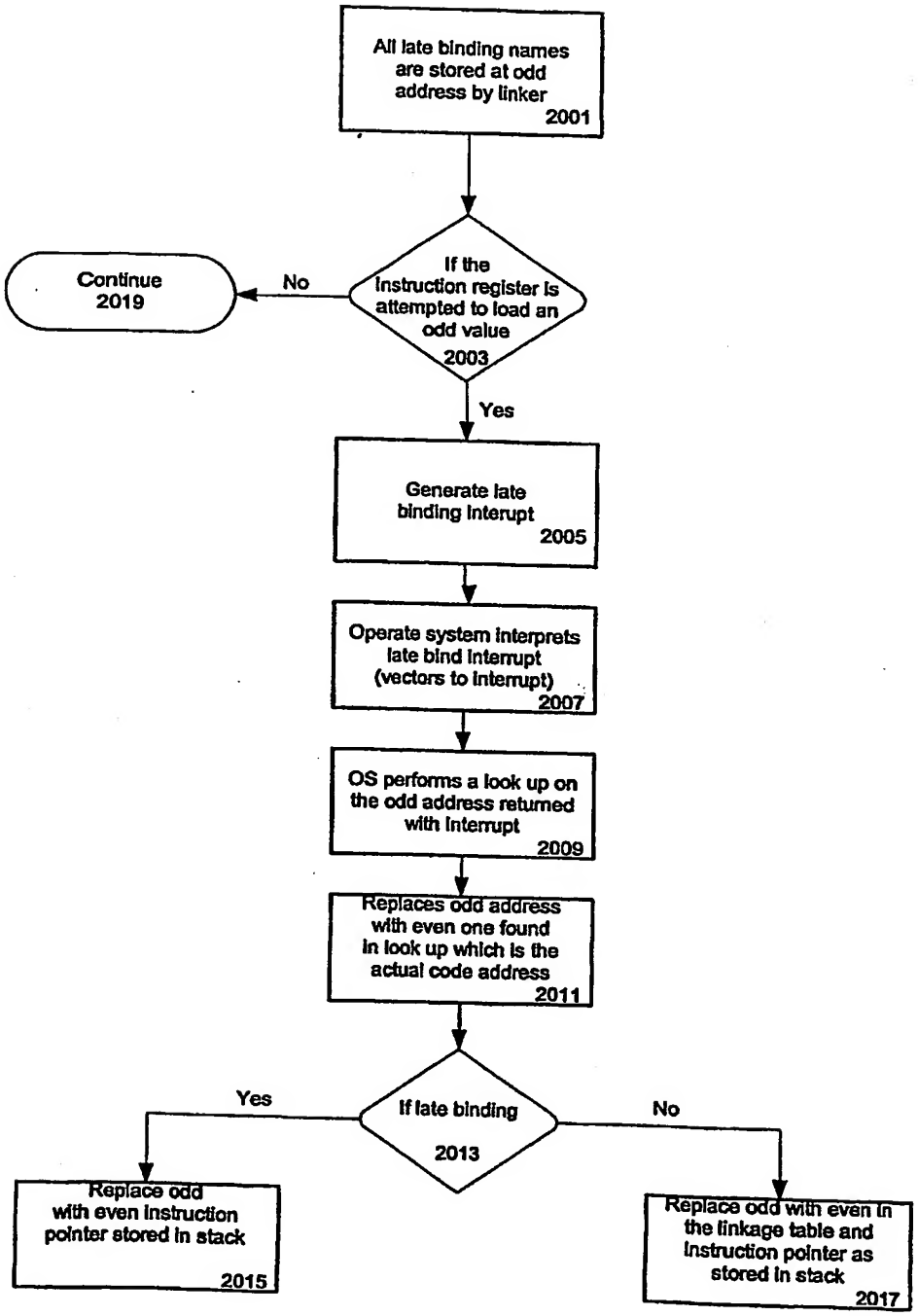

All late binding names are stored at odd address by linker
2001
If the instruction register is attempted to load an odd value
2003
No
Continue
2019
Yes
Generate late binding interupt
2005
Operate system interprets late bind interrupt (vectors to interrupt)
2007
OS performs a look up on the odd address returned with interrupt
2009
Replaces odd address with even one found in look up which is the actual code address
2011
If late binding
2013
Yes
No
Replace odd with even instruction pointer stored in stack
2015
Replace odd with even in the linkage table and instruction pointer as stored in stack
2017

FIG. 20

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/AU03/00994

This Annex lists the known "A" publication level patent family members relating to the patent documents cited in the above-mentioned international search report. The Australian Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

Patent Document Cited in Search Report		Patent Family Member					
US	5522083	US	5613146				
US	5159689	EP	0158320	JP	60214041	JP	60214042
		JP	61049238	JP	61156458	US	4901225
		US	5111388				
US	6035374						
US	5860158						
US	5210828	EP	0376003	JP	2213976		
US	4346435	US	4346436				
EP	0794492	US	5706514				
EP	0385136	JP	2306361	US	5499363		

END OF ANNEX